Copyright Statement

The digital copy of this thesis is protected by the Copyright Act 1994 (New Zealand).

This thesis may be consulted by you, provided you comply with the provisions of the Act and the following conditions of use:

- Any use you make of these documents or images must be for research or private study purposes only, and you may not make them available to any other person.
- Authors control the copyright of their thesis. You will recognize the author's right to be identified as the author of this thesis, and due acknowledgement will be made to the author where appropriate.
- You will obtain the author's permission before publishing any material from their thesis.

General copyright and disclaimer

In addition to the above conditions, authors give their consent for the digital copy of their work to be used subject to the conditions specified on the Library Thesis Consent Form and Deposit Licence.
Robust Data Profiling and Schema Design
for Incomplete Relational Databases

Author: Ziheng WEI
Supervisor: Prof. Sebastian LINK
Co-supervisor: Prof. Bakhadyr KHOUSSAINOV

A thesis submitted in fulfillment of the requirements
for the degree of Doctor of Philosophy
in the
Department of Computer Science
Declaration of Authorship

I, Ziheng Wei, declare that this thesis titled, “Robust Data Profiling and Schema Design for Incomplete Relational Databases” and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.

- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.

- Where I have consulted the published work of others, this is always clearly attributed.

- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.

- I have acknowledged all main sources of help.

- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:  

Date: 07/15/2019
“Le vent se lève / Il faut tenter de vivre.”

“The wind rises / One must try to live.”

Paul Valéry
Abstract

After more than 40 years in use, relational database systems provide a mature and sophisticated technology for effective and efficient data management. For example, SQL is the number one skill that data scientists possess. Relational database systems model a given domain of interest. One of the core problems that still remain is the correct handling of incomplete information. For example, we still do not know how database schemata for data with missing values should be designed. Many approaches to this problem exist, with most depending on a specific interpretation of missing values. Already in classical applications such as accounting, but even more so in modern day applications such as data integration, a reasonable approach should not be dependent on a specific interpretation of missing values. In this thesis, a new approach is established for the design of incomplete relational databases. The new approach is robust under different interpretations of missing values, and only depends on the complete fragments of an incomplete database. The thesis introduces the classes of embedded uniqueness constraints and embedded functional dependencies that enable users to declare completeness and integrity requirements of a given application within a single framework. Embedded functional dependencies are sources of redundant data value occurrences, while embedded uniqueness constraints avoid redundant data value occurrences. This makes them particularly interesting for the design of databases, as data redundancy largely determines how efficiently queries and updates can be processed. We study several computational problems related to the combined class, including axiomatic and algorithmic solutions to their implication problem, structural and computational aspects of Armstrong relations, their discovery problem, and the development of normal forms and decompositions that subsume the classical cases of Boyce-Codd and Third normal forms, respectively, as special cases. Many of our solutions are implemented as prototypes, and theoretical investigations of the computational complexity are complemented by empirical investigations in terms of performance studies on benchmark datasets. As a side result, we also develop a new algorithm that discovers all classical functional dependencies that hold on a given incomplete relation and show that it outperforms state-of-the-art for efficiency, row-, and column-scalability. Overall, the thesis offers a completeness-tailored approach to the design of relational databases.
Acknowledgements

Foremost, I would like to express my sincere gratitude to my supervisor Prof. Sebastian Link for the continuous support of my Ph.D study and related research, for his patience, generosity, motivation, and immense knowledge. His dedication to academia, family and friends always encourages me to pursue a life of perseverance and integrity. I cannot thank him enough for his mentorship.

My sincere thanks also go to Prof. Bakhadyr Khoussainov and Dr. Jiamou Liu. I will be always grateful for their training, knowledge and inspiration. I also gratefully acknowledge the University of Auckland Doctoral Scholarship for supporting my entire study.

To my parents Lihua Wang and Ran Wei, I could not feel luckier to be your son. I love you.
# Contents

Declaration of Authorship iii

Abstract vii

Acknowledgements ix

1 Introduction 1

1.1 Motivation 1

1.2 Research Question, Methodology, and Scope 6

1.3 Contribution 7

1.4 Organization 10

2 Preliminary 11

2.1 Relational Model of Incomplete Data 11

2.2 Classical Data Dependencies 12

2.3 Data Dependency Discovery 16

2.4 Normal Forms 17

2.5 Computational Complexity 18

2.6 Benchmark Datasets 19

2.7 Summary 19

3 Literature Review 21

3.1 Relational Model of Incomplete Data 22

3.1.1 Relational Model 22

3.1.2 Interpretations of Null Markers 23

3.2 Data Dependencies with Incompleteness 24

3.2.1 Keys 24
3.2.2 Functional Dependencies ........................................ 26
3.3 Armstrong Relations .................................................. 27
3.4 Data Dependency Discovery ........................................ 30
  3.4.1 Key and Unique Constraint .................................... 30
  3.4.2 Functional Dependency ......................................... 32
3.5 Normalization ......................................................... 34
3.6 Summary ............................................................... 35

4 Ranked Syntactic and Semantic Data Profiles for Functional Dependencies ............................................... 39
  4.1 Introduction ......................................................... 40
  4.2 Revisiting Previous Algorithms .................................. 45
    4.2.1 Row-efficient Algorithm .................................... 45
    4.2.2 Column-efficient Algorithm ................................ 46
    4.2.3 Hybrid Algorithm ............................................ 47
  4.3 Dynamic Hybrid Algorithm ....................................... 49
    4.3.1 Extended FD-tree ............................................ 51
    4.3.2 Synergized Induction ....................................... 54
    4.3.3 Dynamic Data Manager ..................................... 56
    4.3.4 Validation and Refinement ................................ 57
    4.3.5 When to Update Stripped Partitions ...................... 59
    4.3.6 DHyFD Algorithm ........................................... 61
  4.4 Experiments ......................................................... 65
    4.4.1 Parameter Tuning ............................................ 65
    4.4.2 Performance on Real-world Data ......................... 66
    4.4.3 Scalability Test ............................................. 70
    4.4.4 Memory Consumption ..................................... 75
  4.5 Covers of Syntactic Profiles .................................... 77
  4.6 Relevance Rank of FD ............................................. 80
    4.6.1 Quantitative Analysis .................................... 80
    4.6.2 Qualitative Analysis ..................................... 82
4.7 Armstrong Samples ........................................ 87
4.8 Summary .................................................. 92

5 Embedded Unique Constraints .......................... 95
  5.1 Introduction ............................................. 96
  5.2 Fundamentals of Embedded Unique Constraints ....... 100
  5.3 Armstrong Relations for Embedded Unique Constraints ... 105
  5.4 Experiments ............................................. 113
  5.5 Summary ................................................ 114

6 Discovering Embedded Unique Constraints ............ 117
  6.1 Introduction ............................................. 118
  6.2 Computational Complexity ............................ 123
  6.3 EUC-tree ............................................... 125
  6.4 Column-efficient Discovery ......................... 127
  6.5 Row-efficient Discovery ............................. 133
  6.6 Hybrid Discovery .................................... 138
    6.6.1 Hybrid E-traversal ............................ 142
    6.6.2 Hybrid U-traversal ............................ 144
  6.7 Experiments ........................................... 145
    6.7.1 Performance on Real-world Data ............... 145
    6.7.2 Scalability Test ............................... 146
    6.7.3 Coverage ....................................... 147
  6.8 Query Optimizations with EUCs ..................... 150
    6.8.1 Implementation of EUCs ....................... 151
    6.8.2 Query Optimization Examples ............... 152
  6.9 Armstrong Samples ................................... 156
  6.10 DataProf: a Semantic Profiling Software Tool for Iterative Data Cleansing and Business Rule Acquisition .... 159
    6.10.1 Background .................................... 159
    6.10.2 Overview of Architecture .................... 162
6.10.3 Demonstration Scenario ........................................ 163
6.11 Summary ................................................... 166

7 Data Completeness Tailored Database Design 169
7.1 Introduction .................................................. 170
7.2 Vision and Example ........................................... 172
7.3 Foundations .................................................. 177
  7.3.1 Basics ..................................................... 177
  7.3.2 Axiomatic Characterization ............................... 179
  7.3.3 Algorithmic Characterization ............................. 186
7.4 Normal Forms ................................................ 190
  7.4.1 E-Redundancy Free Normal Form ....................... 190
  7.4.2 E-BCNF .................................................. 192
  7.4.3 E-RFNF .................................................. 195
  7.4.4 Efficient Testing .......................................... 195
  7.4.5 E-3NF ..................................................... 196
  7.4.6 Hardness of Normal Form Criteria ....................... 199
7.5 Tailoring Normalization ....................................... 199
  7.5.1 E-BCNF Decomposition ................................ 200
  7.5.2 E-3NF Synthesis ......................................... 202
7.6 Experiments .................................................. 204
  7.6.1 E-Redundancy and EFD Ranking ......................... 204
  7.6.2 Quality of Decompositions ............................... 205
  7.6.3 Size and Time of Decompositions ....................... 209
  7.6.4 Qualitative Analysis ..................................... 209
7.7 Summary ..................................................... 214

8 Discovering Embedded Functional Dependencies 217
8.1 Introduction .................................................. 218
8.2 Computational Complexity ................................... 221
8.3 EFD-trees .................................................... 223
8.4 Row-efficient Discovery ........................................... 226
8.5 Column-efficient Discovery ....................................... 232
8.6 Hybrid Discovery .................................................. 238
8.7 Experiments ......................................................... 242
  8.7.1 Performance on Real-world Data .............................. 242
  8.7.2 Scalability Test ................................................ 244
8.8 Armstrong Samples ............................................... 245
8.9 Summary ............................................................ 250

9 Conclusion .............................................................. 253
  9.1 Conclusion ......................................................... 253
  9.2 Future Work ....................................................... 259

Bibliography ............................................................ 261
List of Figures

1.1 Effort on tasks in the data science pipeline .......................... 2

2.1 Attribute lattice .......................................................... 17

4.1 The FD-tree (left) and extended FD-tree (right) for FDs $A \rightarrow B$, $AB \rightarrow CD$, and $CD \rightarrow B$ ................................. 47
4.2 Sampling-validation hybridization strategy .............................. 49
4.3 A high level view of DHyFD .............................................. 50
4.4 Example of id assignments by DDM .................................... 56
4.5 An efficiency-inefficiency ratio calculation ............................. 60
4.6 Time of FD discovery on weather (left) and uniprot512k (right) with different efficiency-inefficiency ratios .................. 66
4.7 Best performers on weather datasets ................................... 71
4.8 Best performers on diabetic datasets ................................... 72
4.9 Row scalability test on weather datasets ............................... 72
4.10 Column scalability test on diabetic datasets .......................... 73
4.11 Column scalability test on ncvoter datasets ........................... 74
4.12 Row scalability test on ncvoter datasets ............................... 74
4.13 Memory consumption (MB) of FD discoveries on ncvoter datasets with increasing numbers of rows ...................... 76
4.14 Memory consumption (MB) of FD discoveries on ncvoter datasets with increasing numbers of columns .................... 77
4.15 Balancing time and memory tradeoff with the efficiency-inefficiency ratio on lineitem ................................. 78
4.16 Distributions of canonical FDs’ relevance ranks in complete benchmarks ........................................ 82

4.17 Distributions of canonical FDs’ relevance ranks in incomplete benchmarks (null = null) ........................ 83

4.18 Distributions of canonical FDs’ relevance ranks in incomplete benchmarks (null ≠ null) ....................... 84

4.19 Distributions of canonical FDs’ relevance ranks across increasing row-fragments of ncvoter datasets .......... 86

4.20 Scalability test on sizes of semantic profiles ................................................................................. 91

5.1 Acquisition Framework .................................................................................................................. 105

5.2 Average time of computing Armstrong relations ........................................................................... 113

5.3 Average sizes and average number of null markers in Armstrong relations ....................................... 114

6.1 Example of an eUC-tree .................................................................................................................. 127

6.2 Row scalability on uniprot1k and column scalability on weather .................................................. 148

6.3 Coverage of incomplete datasets .................................................................................................. 149

6.4 Relative scope of individual eUCs ................................................................................................. 149

6.5 Graphical User Interface .............................................................................................................. 163

6.6 Acquisition Framework of DATAPROF ....................................................................................... 163

6.7 Values of name_prefix (left) and name_suffix (right) ............................................................... 165

7.1 Data quality driven schema design .............................................................................................. 173

7.2 Real-world example of data-completeness tailored database schema design ................................. 175

7.3 EFD relevance distributions of benchmark datasets ...................................................................... 206

7.4 EFD relevance distributions of ncvoter datasets varying on different numbers of rows .................. 207

7.5 Average percentage of schemata in E-3NF and E-BCNf by given size of E ........................................ 208
xxi

7.6 Elimination of E-redundancy by E-3NF (vertical left-hand-side),
and E-preservation by E-BCNF (vertical right-hand-side) . . . . 210
7.7 Elimination of E-redundancy by E-3NF (vertical LHS), and Epreservation by E-BCNF (vertical RHS) by row-fragments of
ncvoter . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 211
7.8 Average cardinality of decompositions and time (s) taken to
compute them . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 212
7.9 Relevance eFD distributions of ncvoter1k with two different application requirements . . . . . . . . . . . . . . . . . . . . . . . . 213
7.10 Relevance distribution of eFDs in ncvoter1k for application E1 . 213
7.11 Relevance distribution of eFDs in ncvoter1k for application E2 . 214
7.12 E1 -redundancies caused by eFD full_phone_num: last_name, city,
register_date → zip_code . . . . . . . . . . . . . . . . . . . . . . . . 214
8.1 EFDs: a multidimensional meta view of incomplete data . . . . 220
8.2 Data redundancy eliminated by eFDs. . . . . . . . . . . . . . . . 221
8.3 Representations of eFDs {( AB, A → B), ( AC, A → C )} . . . . . 225
8.4 Left: the attribute lattice of LHSs over R = { A, B, C }. Right: the
attribute lattice (within the parallelogram) of LHS { A}’s embeddings . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 227
8.5 Hybrid discovery algorithms for eFDs. . . . . . . . . . . . . . . 238
8.6 Hybrid e-discovery algorithms for eFDs. . . . . . . . . . . . . . 239
8.7 Time scalability of eFD discovery algorithms . . . . . . . . . . . 246
8.8 Memory scalability of eFD discovery algorithms . . . . . . . . . 247
8.9 Scalability of Armstrong sample sizes. . . . . . . . . . . . . . . . 251


List of Tables

2.1 Benchmark datasets ............................................. 20

4.1 Syntactic and semantic profiles for pdb with over 17 million rows
and 13 columns .................................................. 42
4.2 Toy relation for Example 4.6 .................................. 62
4.3 Runtime (in seconds) of discovering FDs from benchmarks under null = null semantics ....................... 68
4.4 Runtime (in seconds) of discovering FDs from benchmarks under null ≠ null semantics ....................... 69
4.5 Runtime (in seconds) of different implementations of FDEP .............................................. 70
4.6 Memory consumption (MB) of FD discovery on benchmarks under null = null semantics ....................... 75
4.7 Memory consumption (MB) of FD discovery on benchmarks under null ≠ null semantics ....................... 76
4.8 Properties of left-reduced & canonical covers ............... 79
4.9 Data redundancy in numbers and percentages ............... 81
4.10 Snippet of ncvoter to illustrate data redundancy ............... 85
4.11 Semantic Profiles of Benchmark Data ......................... 89

5.1 A relation with no candidate key, no certain key, and no key set ........................................... 98
5.2 Axiomatization B .................................................. 102

6.1 Snippet of the ncvoter1k dataset ............................... 119
6.2 An example for row-efficient eUC discovery ................... 131
6.3 Runtime (in seconds) of the three algorithms to discover eUCs
from incomplete data ........................................... 146
6.4 Runtime (in seconds) of the three algorithms to discover eUCs from complete data ........................................ 146
6.5 Runtime (in seconds) of the three algorithms to discover UCCs from incomplete data ........................................ 147
6.6 Performance measures of SQL Statements in SQL Server .......... 150
6.7 Query optimization: eUC v.s. UC ........................................ 153
6.8 Query optimization: efficient join ........................................ 154
6.9 Query optimization: efficient scan ........................................ 155
6.10 Query optimization: efficient sorting .................................... 156
6.11 Traits of real-world Armstrong samples .............................. 158
6.12 Armstrong sample of bridges with eUCs: (Ø, {0}), (Ø, {2, 3}) and (Ø, {3, 5}). Attributes: identifier (0), river (1), location (2), erected year (3), purpose (4), length (5), lanes (6), clear-g (7), t-or-d (8), material (9), span (10), rel-l (11), and type (12) ............... 159
6.13 Armstrong sample of bridges dataset considering UCCs: {0}, {2, 3}, {3, 4, 5}, {3, 5, 11} and {3, 5, 12} ..................... 159
6.14 eUCs (left): ({6}, {0, 1, 4, 7}), (Ø, {0, 1, 6, 7, 9}), and UCCs (right): {0, 1, 4, 6, 7}, {0, 1, 6, 7, 9} ..................... 160
6.15 Embedded uniqueness constraints using voter_id ................. 165
6.16 Snapshot of Armstrong sample for ncvoter ......................... 165
6.17 Sampled records with suspicious phone numbers ............... 166

7.1 Relation r over R = dflp ........................................ 178
7.2 Axiomatization for eUCs and eFDs .................................. 180
7.3 Inference rules from E ........................................ 185
7.4 Time of computing redundant data value occurrences in benchmarks ........................................ 204

8.1 Runtime (in seconds) of discovering eFDs from benchmark real-world datasets with different algorithms .................. 243
8.2 Memory consumptions (MB) of the eFD discovery algorithms 244
8.3 Time (seconds) of computing eFD Armstrong samples . . . . . 251
List of Definitions

2.1 Relation Schema ................................................. 11
2.2 Tuple ............................................................. 12
2.3 Relation .............................................................. 12
2.4 Complete Tuple ...................................................... 12
2.5 Data Dependency ...................................................... 12
2.6 Size and Cardinality ............................................... 13
2.7 Dependency Implication ............................................. 13
2.8 Cover ................................................................. 13
2.9 Syntactic Derivation ............................................... 13
2.10 Closure ............................................................... 14
2.11 Soundness .......................................................... 14
2.12 Completeness ......................................................... 14
2.13 Key ................................................................. 14
2.14 Functional Dependency ............................................ 15
2.15 Minimal Cover of Keys ............................................ 15
2.16 LHS-reduced Cover of FDs ....................................... 15
2.17 Canonical Cover of FDs ........................................... 15
2.18 Key w.r.t FD ......................................................... 16
2.19 Agree Set ............................................................ 16
2.20 Non-FD ............................................................. 16
2.21 Stripped Partition ................................................... 17
2.22 Value Redundancy .................................................. 17
2.23 RFNF ............................................................... 18
2.24 BCNF ............................................................... 18
2.25 3NF .................................................. 18
2.26 Parameterized Complexity [37] ........................ 18
2.27 Parameterized Reduction[37] .......................... 18
4.1 Extended FD-tree ........................................ 51
4.2 Relevance Rank of FD ................................. 80
5.1 Embedded Unique Constraint .......................... 100
5.2 Armstrong Relations of eUC ............................ 106
6.1 Antecedent Tree ........................................ 125
6.2 EUC-tree ................................................ 126
6.3 Non-eUC ................................................ 128
7.1 Embedded Functional Dependency ....................... 178
7.2 E-closure ............................................... 180
7.3 (E, FD)-Reduct ........................................ 186
7.4 Replacement ............................................ 190
7.5 E-RFNF ............................................... 190
7.6 E-BCNF ............................................... 192
7.7 E-minimal EUCs ....................................... 196
7.8 E-prime Attribute ...................................... 196
7.9 E-3NF ............................................... 197
7.10 E-lossless BCNF Decomposition ...................... 200
7.11 E-dependency-preserving Decomposition .......... 201
7.12 E-3NF Decomposition ................................ 202
8.1 EFD-tree .............................................. 224
8.2 Canonical Cover of EFDs ............................... 226
8.3 Non-eFD ............................................... 232
8.4 Armstrong Sample of EFD ............................. 245
List of Examples

4.1 Extended FD-tree .............................................. 51
4.2 Update FD-tree .............................................. 53
4.3 Synergized Induction of FDs ................................. 54
4.4 Dynamic Data Manager ..................................... 56
4.5 Efficiency-inefficiency Ratio ............................... 60
4.6 DHyFD ......................................................... 63
4.7 Computing Armstrong Relation ............................ 87
5.1 EUC .......................................................... 101
5.2 EUC Derivations ............................................. 102
5.3 Implication of EUCs ......................................... 103
5.4 Implication Algorithm of EUCs ............................ 104
5.5 Armstrong Relation of eUC ................................. 106
5.6 Constructing Armstrong Relation with anti-eUCs ....... 108
5.7 Exponential Size of Armstrong Relation .................. 110
6.1 EUC-tree ...................................................... 126
6.2 Column-efficient eUC Discovery Algorithm ............. 130
6.3 Row-efficient eUC Discovery Algorithm ................ 137
7.1 Implication of EUCs and EFDs ............................ 185
7.2 Implication of EUCs and EFDs ............................ 189
7.3 $E$-redundant ................................................ 191
7.4 $E$-RFNF .................................................... 192
7.5 $E$-BCNF .................................................... 192
7.6 Reduction of $E$-BCNF .................................... 195
7.7 Semantics of $E$-BCNF .................................... 195
7.8 Deciding E-BCNF .............................................. 196
7.9 E-3NF ................................................................. 197
7.10 E-3NF Reduction .................................................. 198
7.11 Deciding E-3NF ..................................................... 199
7.12 E-lossless Decomposition ................................. 201
7.13 E-3NF Synthesis .................................................... 202
8.1 EFD-tree .............................................................. 224
8.2 Derivation of EFD .................................................... 233
8.3 Redundant Non-eFD ................................................... 234
List of Theorems

5.2.1 Theorem (Syntactic Characterization of EUC) ............... 101
5.2.2 Lemma (Total Relation Projection) ....................... 102
5.2.3 Theorem (Axiomatization of EUCs) ....................... 103
5.2.4 Theorem (Complexity of EUC Implication) ............... 104
5.3.1 Theorem (Armstrong Relation of EUC) .................... 106
5.3.2 Theorem (Armstrong Relation of EUC) .................... 107
5.3.3 Lemma (Computing Armstrong Relation of EUC) ........ 108
5.3.4 Theorem (Upper Bound of EUC Armstrong Relations) .... 109
5.3.5 Lemma (Computing Anti-eUCs) ......................... 110
5.3.6 Lemma (Anti-eUC Test) ............................... 111
5.3.7 Theorem (Computing Anti-eUC) ....................... 112

6.2.1 Theorem (NP-completeness of EUC) ....................... 124
6.2.2 Theorem (Fixed-parameter Complexity) .................. 124
6.4.1 Theorem (Non-eUC) .................................. 128
6.4.2 Theorem (Column-efficient eUC Discovery) ............. 131
6.5.1 Proposition (EUC Validation) .......................... 134
6.5.2 Proposition (Existence of EUC) ........................ 134
6.6.1 Theorem (Hybrid EUC Discovery) ..................... 142

7.3.1 Lemma (Soundness of \( \mathcal{E} \)) ....................... 180
7.3.2 Lemma (Extra Rules) ................................ 182
7.3.3 Theorem (Soundness and Completeness of \( \mathcal{E} \)) ... 182
7.3.4 Theorem ((\( E \), FD)-Reduct) ....................... 186
7.3.5 Corollary (Complexity of Embedded Implication) .... 189
<table>
<thead>
<tr>
<th>Section</th>
<th>Theorem/Lemma/Proposition</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.4.1</td>
<td>Theorem (<em>E</em>-RFNF)</td>
<td>191</td>
</tr>
<tr>
<td>7.4.2</td>
<td>Lemma (EUC Reduction)</td>
<td>193</td>
</tr>
<tr>
<td>7.4.3</td>
<td>Theorem (<em>E</em>-BCNF)</td>
<td>194</td>
</tr>
<tr>
<td>7.4.4</td>
<td>Theorem (Semantics of <em>E</em>-BCNF)</td>
<td>195</td>
</tr>
<tr>
<td>7.4.5</td>
<td>Theorem (Deciding <em>E</em>-BCNF)</td>
<td>195</td>
</tr>
<tr>
<td>7.4.6</td>
<td>Lemma (<em>E</em>-prime Attribute)</td>
<td>197</td>
</tr>
<tr>
<td>7.4.7</td>
<td>Theorem (<em>E</em>-3NF Reduction)</td>
<td>197</td>
</tr>
<tr>
<td>7.4.8</td>
<td>Theorem (Deciding <em>E</em>-3NF)</td>
<td>198</td>
</tr>
<tr>
<td>7.4.9</td>
<td>Theorem (NP-completeness)</td>
<td>199</td>
</tr>
<tr>
<td>7.5.1</td>
<td>Theorem (<em>E</em>-lossless Join)</td>
<td>200</td>
</tr>
<tr>
<td>8.2.1</td>
<td>Theorem (NP-completeness of eFD)</td>
<td>222</td>
</tr>
<tr>
<td>8.2.2</td>
<td>Theorem (Fixed-parameter Complexity)</td>
<td>222</td>
</tr>
<tr>
<td>8.4.1</td>
<td>Proposition (Minimal Embedding of EFD)</td>
<td>228</td>
</tr>
<tr>
<td>8.4.2</td>
<td>Proposition (Existence of Embedding)</td>
<td>228</td>
</tr>
<tr>
<td>8.4.3</td>
<td>Theorem (Row-efficient EFD Discovery)</td>
<td>232</td>
</tr>
<tr>
<td>8.5.1</td>
<td>Proposition (Non-EFD)</td>
<td>233</td>
</tr>
<tr>
<td>8.5.2</td>
<td>Proposition (Derivation of EFD)</td>
<td>235</td>
</tr>
<tr>
<td>8.5.3</td>
<td>Theorem (Column-efficient EFD Discovery)</td>
<td>237</td>
</tr>
<tr>
<td>8.8.1</td>
<td>Lemma (Derivation of Non-EFD)</td>
<td>248</td>
</tr>
</tbody>
</table>
List of Algorithms

1. Add FD .............................................. 54
2. Synergized Induction ............................ 55
3. Update Dynamic Data Manager .............. 57
4. FD Validation ..................................... 58
5. Refine Stripped partition ....................... 59
6. DHyFD ............................................ 62
7. Implication of EUCs .............................. 104
8. Computing Armstrong relations .............. 108
9. Compute Anti-eUCs .............................. 112
10. Check Redundancies of eUCs .................. 128
11. Column-efficient EUC Discovery Algorithm .. 130
12. Unique-traversal (Row-efficient Algorithm) . 135
13. E-traversal ...................................... 136
14. Compute Stripped Partition ................. 137
15. Update EUCs .................................... 140
16. Hybrid E-traversal of EUC .................. 143
17. Hybrid U-traversal of EUC .................. 144
18. Compute Armstrong Samples (Direct) ....... 158
19. Decide Implication ............................. 189
20. Check Redundancy of EFD ................... 226
21. LHS-traversal ................................... 229
22. E-traversal ..................................... 230
23. Validation ...................................... 231
24. non-redundant non-eFD ...................... 234
<table>
<thead>
<tr>
<th></th>
<th>Column-efficient algorithm</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td></td>
<td>237</td>
</tr>
<tr>
<td></td>
<td>Completeness-driven Sampling</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
<td>242</td>
</tr>
<tr>
<td></td>
<td>Compute Non-eFDs</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
<td>249</td>
</tr>
</tbody>
</table>
List of Abbreviations

RDBMS  Relational Database Management System
UC    Unique Constraint
FD    Functional Dependency
EUC   Embedded Unique Constraint
EFD   Embedded Functional Dependency
MNEU  Maximal Non-EUC
DDM   Dynamic Data Manager
RFNF  Redundancy Free Normal Form
BCNF  Boyce-Codd Normal Form
3NF   Third (3) Normal Form
E-RFNF Embedded Redundancy Free Normal Form
E-BCNF Embedded Boyce-Codd Normal Form
E-3NF  Embedded Third (3) Normal Form
Chapter 1

Introduction

The introductory chapter contains a section on the motivation of the research and its main ideas, a section on the research question and methodology, a section that lists the main contributions of the research, and a concluding section on the organization of the thesis.

1.1 Motivation

The big data era has lifted the use of relational databases to a new level. Big data is often characterized by the four most popular dimensions of Variety, Velocity, Veracity, and Volume [16]. As we have started to realize that big data is not synonymous with big insight, the area of data science has received more and more attention in academia, industry, and society [14], [16], [54]. The aim of data science is to realize the biggest promise of big data: Value [14], [53], [54]. As illustrated in Figure 1.1, data science follows a value creation chain from data gathering, to data preparation, to data analytics, to data visualization. Most of research and development are focused on analytics and visualization, yet these are the areas that most data scientists spend least of their time. Instead, they spend most of their time on data gathering and data preparation, getting the right data, cleaning it, and putting it in the right shape for analytics. This makes perfect sense for at least two reasons.

1) No matter how good the analytics, if the input data is garbage, the result of the analytics will be garbage, too.\footnote{https://bit.ly/2i00rdL} 2) While analytics and visualization
are mature areas of research with a lot of tools and technologies, data gathering and preparation are arguably subject to fewer scientific foundations that have resulted in useful tools. Alongside Python and R, SQL is considered to be one of the top three skills of data scientists\(^2\).

The rich landscape of big data applications is a strong driver for the use of mature relational database technology, but also requires careful thoughts on expanding the technology to serve the needs of the new applications. A barrier to the development of data management and reporting applications is the current lack of opportunity to embed data quality requirements of the application in the development process. Indeed, creating value from data means is only possible if the input data meets the requirements of applications. Popular data quality dimensions include data integrity, data consistency, data completeness, and data timeliness. It is therefore important that frameworks and tools are developed that enable users to declare these requirements and lift them to first-class citizens in the development process.

Among these data quality dimensions, data completeness is of the most interest for my research. In the relational model of data, in particular in its practical use in SQL, missing data values are uniformly represented by the special symbol \(\bot\), called the null marker. Due to popular demand in practical applications, there have been extensive studies to extend the relational model of data with null markers. However, only few of them can potentially be put into practice due to theoretical and computational difficulties.

\(^2\)https://bit.ly/2GHakKW
In previous research, most studies have heavily focused on different interpretation of null markers, including ‘exists but unknown’, ‘inapplicable’, and ‘no information’. The difficulty is that such interpretations must be applied uniformly to all occurrences of the null marker, which is difficult to justify for many applications. This is particularly true applications such as data integration, where data comes from different sources. In modern day practice, applications seem to take a different angle. In exploratory data analysis (EDA), for example, missing data is imputed with statistical methods dependant on subsequent machine learning models. Instead of caring about the actual interpretation of null markers, data analysts tend to experiment with a range of statistics to impute values for null markers, such that the subsequent analytical models can produce an optimized expected value. In relational database management systems (RDBMS), such as Microsoft SQL Server, unique constraints are implemented to uniquely identify records in the presence of missing values. In fact, unique constraints are a part of the SQL standard which enforce uniqueness only for those records that have no missing value on any of the attributes specified by the unique constraint. This illustrates that the nature of missing data is rather diverse and their usage are dependent on the purpose of an application. In a nutshell, interpretations of null markers should remain open. Otherwise, restrictions on the interpretation will greatly diminish the potential of data applications, such as analytics and management. Considering the emerging demand from modern data applications but also the open challenges on treating missing values in classical applications such as query processing, database theory and practice are still lacking a robust approach to managing incomplete data.

The overall goal of my research is to establish new classes of database constraints that enable users to effectively stipulate data quality requirements and be used efficiently for various data management tasks. In this thesis, I
will make a start to this research by considering data integrity and completeness requirements for the tasks of data profiling and schema design. The approach is robust under different interpretations of null marker occurrences, as the semantics of the constraints is solely determined by the complete fragments embedded in incomplete data. In this sense, I let the complete data speak for itself and not be concerned with guesswork around the right meaning of null marker occurrences.

As a starting point the new approach will be applied to the most popular classes of uni-relational constraints, namely unique constraints and functional dependencies. On the one hand, their popularity is founded in Codd’s principle of entity integrity. Unique constraints are used by data management systems to provide efficient access to data. Indeed, the unique constraints can be indexed, which results in the efficient evaluation of many queries. Functional dependencies can express a wide variety of important application semantics, and constitute a rich source of data redundancy. In turn, unique constraints prohibit data redundancy, so the combined class of unique constraints and functional dependencies is important in schema design where the goal is to identify a schema in which any future occurrences of data redundancy in instances over the schema are minimized if not even eliminated. Such schema designs are important in Online Transaction Processing (OLTP) where the minimization of data redundancy can guarantee the efficient processing of updates. Based on the intention to embed data quality requirements in database constraints, the thesis proposes the new notions of embedded unique constraints (eUCs) and embedded functional dependencies (eFDs). These constraint have the form $E : \sigma$ where $E$ denotes a set of columns of the underlying schema, and $\sigma$ is a traditional unique constraint or functional dependency. The main idea of the embedded constraints is to stipulate the traditional semantics of $\sigma$ on the fragment of input tuples that have no missing data values on any of the columns in $E$. As an example, an SQL unique constraint is an embedded unique constraint of the form $E : E$ saying
that there cannot be two different \( E \)-complete tuples that have matching values on all the attributes in \( E \). In contrast, an eUC \( E : U \) with \( U \subseteq E \) provides the opportunity that \( E \)-complete tuples can already be distinguished by the attribute set \( U \) rather than the full attribute set \( E \).

For data profiling, in particular, the new approach is able to discover all those eUCs and eFDs that hold on the complete fragments of the potentially incomplete input data. The novelty over previous work results from the strong separation of completeness requirements from uniqueness and functional dependence requirements, respectively. We illustrate the applications of these new discovery algorithms to the ranking of the constraints, and the detection of dirty, corrupt or inconsistent data caused by null marker occurrences.

For relational schema design, the new approach lifts completeness requirements to first-class citizens of relational database schema design. Depending on the completeness and functional dependence requirements that users stipulate for the target applications, the new framework tailors classical relational schema design in terms of Boyce-Codd and Third Normal Forms to these requirements. Indeed, different relational designs can be computed based on different completeness requirements, and any future data that will be stored for those schemata will meet the requirements by design. Data that does not meet the requirements will not be used for the applications, but can still be stored in a data lake for future analysis. For example, business keys can be enforced on application data without the need to clean up any null marker occurrences. This de-couples the data cleaning process for the incomplete data from any applications.
1.2 Research Question, Methodology, and Scope

Based on our motivation from the previous section the research presented in this thesis aims at contributions to the following overarching research question:

What are notions of uniqueness constraints and functional dependencies that are robust under different interpretations of missing values, allow users to stipulate data completeness requirements, and permit efficient solutions to important computational problems in data profiling and relational schema design?

The computational problems we consider in this work are quite extensive, and include the following:

- the axiomatic and algorithmic characterizations of the implication problem associated with eUCs and eFDs,
- structural and computational properties of Armstrong relations associated with eUCs and eFDs,
- the discovery problem associated with eUCs and eFDs,
- the presentation of the output of discovery algorithms in terms of covers of the sets of constraints that hold and a semantic sample that satisfies the same constraints as the input dataset,
- the formalization of semantic properties desirable for relation schemata to be considered as well-designed with respect to completeness requirements, and their syntactic characterization in terms of eUCs and eFDs,
- the computation of well-designed schemata from one that are not.

Each of these problems will be defined precisely, and solutions will be developed that involve mathematical proofs of correctness. As examples, these include the development of algorithms that are shown to be correct and the proof that a problem is (hard for all problem) in a specific complexity class. In addition, most of the algorithms are implemented in prototype systems and
applied to real-world benchmark datasets. Extensive performance analysis have been carried out to complement the theoretical complexity analysis by some insight into the actual time and space it takes on actual data, but also to illustrate the usefulness of our concepts in concrete application scenarios.

We emphasize that the research opens up various avenues to extend the work in this thesis. For instance, it is desirable to extend our approach to other data quality dimensions, including the accuracy and timeliness dimension. It is also desirable to look at other classes of constraints on other models of data, such as multivalued dependencies and inclusion dependencies on models of Web data or uncertain data. This, however, falls outside the scope of this thesis.

1.3 Contribution

Overall, the research findings establish a completeness-tailored approach to the design of relational databases. This overall contribution is the result of several contributions that can be summarized as follows. For terminology that the reader is not yet familiar, we refer to subsequent chapters in which these concepts will be defined in abstract terms and by illustrative examples.

- We propose the novel classes of embedded unique constraints (eUCs) and embedded functional dependencies (eFDs) over incomplete relational databases.

- We establish sound and complete axiomatizations for the implication problem associated with the individual and combined classes of eUCs and eFDs.

- We establish linear-time decision algorithms for the implication problem associated with the individual and combined classes of eUCs and eFDs.

- We investigate structural and computational properties of Armstrong relations for eUCs and eFDs.
• We improve the benchmark discovery algorithm for the class of classical functional dependencies (FDs) in terms of efficiency, row, and column scalability, the output size, and the ranking of the output.

• We prove that the decision variant of the discovery problem associated eUCs and eFD is \( \text{NP}\)-complete and \( \text{W}[2]\)-complete.

• We propose the first column-efficient, row-efficient, and hybrid discovery algorithms for the classes of eUCs and eFDs.

• We conduct extensive experiments with benchmark data that demonstrate the efficiency and scalability of the proposed eUC and eFD discovery algorithms in terms of runtime and memory consumption.

• We show how eUCs can be specified and enforced in SQL Server using filtered indices and views.

• We use real-world examples to illustrate how the enforcement of entity and referential integrity, and how the performance of important types of queries can be optimized by enforcing eUCs, and how these improve in comparison to enforcing unique constraints.

• We propose the computation of Armstrong samples for eUCs, eFDs and FDs as an alternative and more user-friendly representation of the output of discovery algorithms. Our experiments illustrate that Armstrong samples are very useful because of their size is typically much smaller than that of the input data.

• We implement a fully-fledged software called DataProf, which is used to demonstrate a real-world data cleaning application.

• We propose the ranking of FDs and eFDs by the number of data redundancies they cause. This is useful for acquiring meaningful business rules, and focusing the attention of database designers on the constraints that are more relevant for the given dataset.
• We propose a conceptual framework for data-quality driven database schema design, using applications requirements to generate schemata that are fit with purpose.

• We define the $E$-Redundancy Free Normal Form (E-RFNF) and characterize $E$-RFNF syntactically by an extension of the well-known Boyce-Codd Normal Form (BCNF), called $E$-BCNF.

• We provide a suitable extension of the well-known Third Normal Form (3NF), called $E$-3NF.

• We show how to transform any given set of eUCs and eFDs into a set of classical FDs such that classical lossless BCNF-decomposition and lossless 3NF-synthesis algorithms result in schemata that are in $E$-BCNF and $E$-3NF, respectively.

• We conduct comprehensive experiments on real-world benchmark data and schemata that show the effectiveness and efficiency of our proposed framework, but also quantify insights on data redundancy and classical normalization in unprecedented form.
1.4 Organization

In Chapter 2, we will define the necessary preliminary concepts that are required for the development of our ideas. In Chapter 3, we will discuss related previous research. In Chapter 4, we establish new contributions to the state-of-the-art discovery algorithm for classical functional dependencies. We give also first insights on how to use functional dependencies to generate syntactic and semantic profiles for large incomplete data. In Chapter 5, we will introduce embedded unique constraints (eUCs) to generalize classical keys and SQL unique constraints. We will identify cases for their use to establish entity integrity, referential integrity, and query optimization. We will then establish axiomatic and algorithmic characterizations of their implication problem, and will also develop structural and algorithmic characterizations of Armstrong relations for eUCs. In Chapter 6, we will study the discovery problem of eUCs, as well as the computation of Armstrong samples. We will carry out real-world case studies to demonstrate our data profiling software – DataProf. In Chapter 7, we investigate the combined class of eUCs and eFDs for the purposes of relational database schema design. More specifically, we will formalize a completeness-driven design theory consisting of E-RFNF, E-BCNF and E-3NF. In Chapter 8, we will study the discovery problem of embedded functional dependencies (eFD), as well as the computation of Armstrong samples. Lastly, we conclude all our research results in Chapter 9.
Chapter 2

Preliminary

In this chapter, we will introduce the preliminary concepts and notions that we will use to develop the theories around *embedded unique constraints* (eUCs) and *embedded functional dependencies* (eFDs) as well as their applications. More specifically, Section 2.1 will recall concepts of the relational model of data and its extensions to capture incomplete data, Section 2.2 will recall the notions associated with keys, functional dependencies, and their associated implication problem, Section 2.3 will recall the definitions surrounding the discovery problem, Section 2.4 will recall the notions around classical normal forms, Section 2.5 will recall the notions around classical and parameterized complexity classes, and Section 2.6 will introduce the main real-world benchmark datasets we will use in this thesis. The chapter concludes in Section 2.7 with a brief summary.

2.1 Relational Model of Incomplete Data

Let $\mathbb{A} = \{A_1, A_2, \ldots\}$ be a countable and infinite set of distinct symbols, called *attributes* or *columns*.

**Definition 2.1 (Relation Schema)** A relation schema is a finite, non-empty set of attributes, normally denoted as $R$. Each attribute $A \in \mathbb{A}$ is associated with a domain $\text{dom}(A)$. 

\[
\]


To distinguish missing values, we assume that the domain of every attribute contains a distinguished symbol called null marker, which we denote it by $\bot$. We emphasize that $\bot$ is not a domain value but a marker because the interpretation of a null maker varies on different applications.

**Definition 2.2 (Tuple)** A tuple (a.k.a a row or a record) $t$ over $R$ is a function which maps each $A \in R$ to a value in $\text{dom}(A)$, namely $t(A) \in \text{dom}(A)$.

**Definition 2.3 (Relation)** A relation $r$ over $R$ is a finite set of tuples over $R$. The size of the relation, denoted as $|r|$ is the number of tuples the relation contains.

Let $X = \{A_1, A_2, \cdots, A_m\}$ be a set of attributes. For simplicity, we sometimes write $X$ as $A_1 A_2 \cdots A_m$, and the union of $X$ and another attribute set $Y$ as $XY$. Let $X \subseteq R$. For a tuple $t$ over $R$, we use the notation $t(X)$ to denote the projection of $t$ onto $X$. To stipulate completeness, we define total tuples for relational model of incomplete data.

**Definition 2.4 (Complete Tuple)** A tuple $t$ over $R$ is $X$-total or $X$-complete if and only if $t(A) \neq \bot$ for all $A \in X$.

Furthermore, we use $r^X$ to denote $\{t \in r \mid t$ is $X$-total$\}$.

### 2.2 Classical Data Dependencies

**Definition 2.5 (Data Dependency)** A data dependency of class $C$ is a statement which enforces semantic properties on a given collection of data.

Let $\Sigma \cup \{\phi\}$ be a finite set of data dependencies of class $C$ over relation schema $R$. 
**Definition 2.6 (Size and Cardinality)** The size of $\Sigma$, denoted by $|\Sigma|$, is the number of data dependencies in $\Sigma$. The cardinality of $\Sigma$, denoted by $||\Sigma||$, is the total number of attribute occurrences in $\Sigma$. 

The $C$-implication problem is to decide whether for the arbitrarily given relation schema $R$, and the arbitrarily given set $\Sigma \cup \{\varphi\}$ of data dependencies from class $C$ on $R$, $\Sigma$ implies $\varphi$.

**Definition 2.7 (Dependency Implication)** The set of data dependencies $\Sigma$ implies data dependency $\varphi$, denoted by $\Sigma \models \varphi$, if and only if every relation over $R$ that satisfies all the elements in $\Sigma$ also satisfies $\varphi$.

The implications of $\Sigma$ over class $C$ are used to define a better representation of $\Sigma$.

**Definition 2.8 (Cover)** A set of data dependencies $\Sigma'$ of class $C$ over $R$ is a cover of $\Sigma$ if and only if $\Sigma' \models \varphi$ for every data dependency $\varphi$ where $\Sigma \models \varphi$. $\Sigma'$ is a non-redundant cover of $\Sigma$ if and only if $\Sigma' - \{\varphi\}$ does not imply $\varphi$ for all $\varphi \in \Sigma$.

Solving implication problems relies on establishing a set of inference rules that allows to mechanically derive $\varphi$ from $\Sigma$. We apply inference rules of the form

\[
\begin{array}{c}
\text{premises} \\
\hline
\text{conclusion}
\end{array}
\]

**Definition 2.9 (Syntactic Derivation)** Let $\mathcal{R}$ be a set of inference rules over class $C$. Data dependency $\varphi$ is derivable from $\Sigma$ with respect to $\mathcal{R}$, denoted by $\Sigma \vdash_{\mathcal{R}} \varphi$, if and only if there is some finite sequence $\sigma_1, \sigma_2, \ldots, \sigma_n$ such that $\sigma_n = \varphi$ and for every $i < n$, $\sigma_i \in \Sigma$ or $\sigma_i$ results from the conclusion of some inference rule in $\mathcal{R}$ with $\sigma_1, \ldots, \sigma_{i-1}$ as premises.
The notion of closure is often used to solve an implication problem. Let \( \Sigma \) be a set of data dependencies of class \( C \) over relation schema \( R \). \( \mathcal{R} \) is a set of inference rules of \( C \)-dependencies.

**Definition 2.10 (Closure)** The syntactic closure of \( \Sigma \) is \( \Sigma^{+}_{\mathcal{R}} = \{ \Sigma \vdash_{\mathcal{R}} \sigma \mid \sigma \text{ is a } C \text{-dependency over } R \} \); the semantic closure of \( \Sigma \) is \( \Sigma^{*}_{\mathcal{R}} = \{ \Sigma \models_{\mathcal{R}} \sigma \mid \sigma \text{ is a } C \text{-dependency over } R \} \).

The closure property can further characterize the inference rules of data dependencies of class \( C \).

**Definition 2.11 (Soundness)** For any \( C \)-dependencies \( \Sigma \) over \( R \), inference rules \( \mathcal{R} \) is sound if and only if \( \Sigma^{+}_{\mathcal{R}} \subseteq \Sigma^{*}_{\mathcal{R}} \).

**Definition 2.12 (Completeness)** For any \( C \)-dependencies \( \Sigma \) over \( R \), inference rules \( \mathcal{R} \) is complete if and only if \( \Sigma^{*}_{\mathcal{R}} \subseteq \Sigma^{+}_{\mathcal{R}} \).

In fact, sound and complete inference rules are essential tools for solving implications problems of data dependencies.

After being able to decide the implication problem, we can easily compute a non-redundant cover for \( \Sigma \) by successively checking for all \( \varphi \in \Sigma \) whether \( \Sigma - \{ \varphi \} \) implies \( \varphi \) and removing \( \varphi \) from \( \Sigma \) whenever that is the case. In practice, checking whether all the constraints in a non-redundant cover are satisfied by a relation ensures that the overhead of constraint validation is minimized.

Next, we give classical definitions of keys and functional dependencies.

**Definition 2.13 (Key)** A key (superkey) \( K \) over relation schema \( R \) is a subset of \( R \). A relation \( r \) over \( R \) satisfies \( K \), denoted by \( r \models K \), if and only if \( t_{1}(K) = t_{2}(K) \) implies \( t_{1} = t_{2} \) for all \( t_{1}, t_{2} \in r \). We say a key is minimal with respect to \( r \) if and
only if \( r \not\models K - \{ A \} \) for all \( A \in K \).

Note that in this thesis we use key and superkey as synonyms for simplicity. The definition of key may differ from different literature. Also, the minimality of a key is often defined with respect to a set of keys or functional dependencies in the literature. Definition 2.13 is more related to the research problems of this thesis.

**Definition 2.14 (Functional Dependency)** A functional dependency (FD) over relation schema \( R \) is in the form of \( X \rightarrow Y \) where \( X, Y \subseteq R \). A relation \( r \) over \( R \) satisfies FD \( X \rightarrow Y \), denoted by \( r \models X \rightarrow Y \), if and only if \( t_1(X) = t_2(X) \) implies \( t_1(Y) = t_2(Y) \). We say FD \( X \rightarrow Y \) is trivial if \( Y \subseteq X \). We call \( X \) and \( Y \) the left-hand-side (LHS) and right-hand-side (RHS) of the FD respectively.

While investigating the discovery problem of keys and uniqueness constraints, the minimal cover is often used. On the other hand, canonical cover and LHS-reduced cover are often used to reason and discover FDs.

**Definition 2.15 (Minimal Cover of Keys)** Let \( \Sigma \) be a set of keys over relation schema \( R \). The minimal cover of \( \Sigma \) is the set \( \{ K \in \Sigma \mid \forall K' \in \Sigma : K' \not\subseteq K \} \).

**Definition 2.16 (LHS-reduced Cover of FDs)** Let \( \Sigma \) be a set of FDs over relation schema \( R \). The LHS-reduced cover of \( \Sigma \) is the set \( \{ X \rightarrow A \mid X \rightarrow A \in \Sigma, \neg\exists X' \rightarrow A \in \Sigma : X' \subseteq X \} \).

**Definition 2.17 (Canonical Cover of FDs)** Let \( \Sigma \) be a set of FDs over relation schema \( R \). The canonical cover of \( \Sigma \) is the set \( \{ X \rightarrow Y \in \Sigma \mid \Sigma - \{ X \rightarrow Y \} \not\models X \rightarrow Y \} \).

In database design theory, keys also can be defined with respect to a set of
FDs since FDs generalize keys when the relational model of complete data is assumed.

**Definition 2.18 (Key w.r.t FD)** A set of attributes \( K \subseteq R \) is a key (superkey) with respect to a set FDs \( \Sigma \) over \( R \) if and only if \( \Sigma \models K \rightarrow R \). We say key \( K \) is minimal with respect to \( \Sigma \) if and only if \( \Sigma \not\models K - \{A\} \rightarrow R \) for all \( A \in R \). An attribute in \( R \) is prime with respect to \( \Sigma \) if and only if it belongs to some minimal key of \( \Sigma \). □

### 2.3 Data Dependency Discovery

Given a relation, all the data dependencies such as functional dependencies that are satisfied by the relation can be discovered. In this section, we define some important notions used in key and FD discovery.

One of the important ways to model the search space of keys and FDs is to use the attribute lattice. As illustrated in Figure 2.1, an attribute lattice over a relation schema divides the power set of the schema into different levels. At each level, all attribute sets must have the same cardinality.

Unlike constructing a search space like the attribute lattice, notions including agree set and non-FDs allow to compute true keys or FDs in a more inductive manner.

**Definition 2.19 (Agree Set)** Let \( t_1, t_2 \) be tuples in relation \( r \) over relation schema \( R \). The agree set of \( t_1, t_2 \) is \( \text{agr}(t_1, t_2) = \{ A \in R \mid t_1(A) = t_2(A) \} \). We say \( t_1, t_2 \) agree on the set \( \text{agr}(t_1, t_2) \). □

**Definition 2.20 (Non-FD)** A non-FD over relation schema \( R \) is in the form of \( X \not\rightarrow Y \) where \( X, Y \subseteq R \). A relation \( r \) over \( R \) satisfies non-FD \( X \not\rightarrow Y \) if and only if there exist \( t_1, t_2 \in r \) where \( t_1(X) = t_2(X) \) and \( t_1(A) \neq t_2(A) \) for all \( A \in Y \). □
2.4 Normal Forms

_Stripped partitions_ also play an important role in help of discovering keys and functional dependencies.

**Definition 2.21 (Stripped Partition)** Let $r$ be a relation over relation schema $R$ and $X \subseteq R$. The $X$-equivalence class of tuple $t \in r$ is the set $[t]_X = \{s \in r | s(X) = t(X)\}$. The stripped partition of $r$ over $X$ is $\pi_X(r) = \{[t]_X | t \in r, |[t]_X| \geq 2\}$.

Note that sometimes we only write $\pi_X$ instead of $\pi_X(r)$ if $r$ is clear from the context.

### 2.4 Normal Forms

In Chapter 7, we extend classical database design theory to relational model of incomplete data. Particularly, _Redundancy-free normal forms_ (RFNF), _Boyce-Codd normal forms_ (BCNF), and _Third normal forms_ (3NF) are of our most interests [69].

**Definition 2.22 (Value Redundancy)** Let $r$ be a relation and a set $\Sigma$ of functional dependencies over a relation schema $R$. The occurrence of a value $t(A)$, where $A \in R$ and $t \in r$, is redundant in $r$ with respect to $\Sigma$ if for every replacement of $t(A)$ by...
a distinct value $v$ in $\text{dom}(A)$ such that $v \neq t(A)$, result in the new relation $r'$, we have that $r'$ does not satisfy $\Sigma$.

**Definition 2.23 (RFNF)** A relation schema $R$ is in Redundancy Free Normal Form with respect to a set $\Sigma$ of FDs over $R$ if there exists no relation $r$ over $R$ and a value occurrence that is redundant in $r$ with respect to $\Sigma$.

**Definition 2.24 (BCNF)** A relation schema $R$ is in Boyce-Codd Normal Form with respect to a set $\Sigma$ of FDs over $R$ if for every non-trivial FD $X \rightarrow A \in \Sigma$, $X$ is a superkey for $R$ with respect to $\Sigma$.

**Definition 2.25 (3NF)** A relation schema $R$ is in Third Normal Form (3NF) with respect to a set of FDs over $R$ if for every non-trivial FD $X \rightarrow Y \in \Sigma$, either $X$ is a superkey for $R$ with respect to $\Sigma$ or $A$ is a prime attribute with respect to $\Sigma$.

### 2.5 Computational Complexity

In Chapter 6 and 8, we discuss parameterized time-complexity of the discovery problems with incomplete data.

**Definition 2.26 (Parameterized Complexity [37])** Let $I$ be an instance of decision problems of class $\mathcal{D}$ and a natural number parameter $k$. $(I, k)$ denotes the corresponding instance of fixed-parameter problem of $\mathcal{D}$. The fixed-parameter problem of class $\mathcal{D}$ is fixed-parameter tractable if a given instance $(I, k)$ can be solved in time $O(f(k) \cdot p(|I|))$ where $p$ is a polynomial function.

We use FPT to denote the class of fixed-parameter tractable problems. Let $\mathcal{D}$ and $\mathcal{D'}$ be two classes of parameterized problem.

**Definition 2.27 (Parameterized Reduction [37])** A parameterized reduction
from $\mathcal{D}$ to $\mathcal{D}'$ is a fixed-parameter tractable algorithm that computes a corresponding instance of $\mathcal{D}'$ for any given instance of class $\mathcal{D}$ and the parameters of matching instances are only dependent on each other.

We use $\mathcal{D} \preceq_{\text{FPT}} \mathcal{D}'$ to denote that there is a fixed-parameter reduction from $\mathcal{D}$ to $\mathcal{D}'$. Moreover, we say $\mathcal{D}$ and $\mathcal{D}'$ are FPT-equivalent if $\mathcal{D} \preceq_{\text{FPT}} \mathcal{D}'$ and $\mathcal{D}' \preceq_{\text{FPT}} \mathcal{D}$. Parameterized problems are divided into a hierarchy of complete complexity classes by the problem WEIGHTED t-NORMALIZED SATISFIABILITY [37] i.e. $\text{FPT} \subseteq \mathcal{W}[1] \subseteq \mathcal{W}[2] \ldots$.

### 2.6 Benchmark Datasets

We use a collection of benchmark datasets across this thesis, which are widely used in the previous research [41], [52], [87], [88], [90]. The datasets only contain data of relational model and will be treated as relations. For each dataset we show its general information in Table 2.1 including alias, number of rows (#Rows), number of columns (#Columns), number of missing values (#⊥), and data source. These datasets help us test the performance of our applications in data dependency discovery, semantic sampling, data cleaning, and database design. They also allow us to conduct case studies on real-world scenarios and demonstrate practicality of our proposed concepts.

### 2.7 Summary

This chapter serves as a reference point for all the necessary concepts that are required for the development of our ideas in the thesis. These include the data model, the data dependencies, the computational problems of discovery, implication, and schema design, as well as the benchmark datasets that will be used throughout the experimental evaluation of our concepts.
## Table 2.1: Benchmark datasets

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Alias</th>
<th>#Rows</th>
<th>#Columns</th>
<th>#⊥</th>
<th>Data Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iris</td>
<td>iris</td>
<td>150</td>
<td>5</td>
<td>0</td>
<td>[10]</td>
</tr>
<tr>
<td>Balance-scale</td>
<td>balance</td>
<td>625</td>
<td>5</td>
<td>0</td>
<td>[10]</td>
</tr>
<tr>
<td>Abalone</td>
<td>abalone</td>
<td>4177</td>
<td>9</td>
<td>0</td>
<td>[10]</td>
</tr>
<tr>
<td>Nursery</td>
<td>nursery</td>
<td>12960</td>
<td>9</td>
<td>0</td>
<td>[10]</td>
</tr>
<tr>
<td>Letter</td>
<td>letter</td>
<td>20000</td>
<td>17</td>
<td>0</td>
<td>[10]</td>
</tr>
<tr>
<td>Chess</td>
<td>chess</td>
<td>28056</td>
<td>7</td>
<td>0</td>
<td>[10]</td>
</tr>
<tr>
<td>Adult</td>
<td>adult</td>
<td>48842</td>
<td>14</td>
<td>0</td>
<td>[10]</td>
</tr>
<tr>
<td>Fd-reduced-30</td>
<td>fd</td>
<td>250000</td>
<td>30</td>
<td>0</td>
<td>[90]</td>
</tr>
<tr>
<td>Lineitem</td>
<td>lineitem</td>
<td>6001215</td>
<td>16</td>
<td>0</td>
<td>[90]</td>
</tr>
<tr>
<td>Breast-cancer-wisconsin</td>
<td>cancer</td>
<td>699</td>
<td>11</td>
<td>16</td>
<td>[10]</td>
</tr>
<tr>
<td>Bridges</td>
<td>bridges</td>
<td>108</td>
<td>13</td>
<td>77</td>
<td>[10]</td>
</tr>
<tr>
<td>Echocardiogram</td>
<td>echo</td>
<td>132</td>
<td>13</td>
<td>132</td>
<td>[10]</td>
</tr>
<tr>
<td>Hepatitis</td>
<td>hepatitis</td>
<td>155</td>
<td>20</td>
<td>167</td>
<td>[10]</td>
</tr>
<tr>
<td>Horse</td>
<td>horse</td>
<td>368</td>
<td>28</td>
<td>1605</td>
<td>[10]</td>
</tr>
<tr>
<td>Ncvoter_1001r_19c</td>
<td>ncvoter1k</td>
<td>1000</td>
<td>19</td>
<td>2863</td>
<td>[90]</td>
</tr>
<tr>
<td>Plista_1k</td>
<td>plista</td>
<td>1000</td>
<td>63</td>
<td>23317</td>
<td>[90]</td>
</tr>
<tr>
<td>Flight_1k</td>
<td>flight</td>
<td>1000</td>
<td>109</td>
<td>51938</td>
<td>[90]</td>
</tr>
<tr>
<td>Uniprot_1001r_223c</td>
<td>uniprot1k</td>
<td>1000</td>
<td>223</td>
<td>179129</td>
<td>[90]</td>
</tr>
<tr>
<td>Diabetic_30c</td>
<td>diabetic</td>
<td>101766</td>
<td>30</td>
<td>192849</td>
<td>[10]</td>
</tr>
<tr>
<td>China_weather_pm2.5</td>
<td>weather</td>
<td>262920</td>
<td>18</td>
<td>418580</td>
<td>[10]</td>
</tr>
<tr>
<td>PDBX_POLY_SEQ_SCHEME</td>
<td>pdb</td>
<td>17305799</td>
<td>13</td>
<td>2035242</td>
<td>[90]</td>
</tr>
<tr>
<td>Uniprot_512001r_30c</td>
<td>uniprot512k</td>
<td>512000</td>
<td>30</td>
<td>3759296</td>
<td>[90]</td>
</tr>
</tbody>
</table>
Chapter 3

Literature Review

Embedded unique constraints (eUCs) and embedded functional dependencies (eFDs) are two novel classes of data dependencies. They extend the classical classes of keys and functional dependencies, respectively, from the relational model without missing data to the relational model with incomplete data. The semantics of both classes is defined independently of any occurrences of missing data, and therefore only relies on complete fragments of the given incomplete datasets. This is a core requirement for our approach in order to develop robust methods for data profiling and schema design. For the scope of this thesis, we are interested in the axiomatization, implication, discovery, and normalization theories of each of the two classes in isolation but also their combination. On a practical side, we aim at building a metadata management framework for data profiling, data cleaning, and database schema design by utilizing tools including discovery algorithms and Armstrong samples of eUCs and eFDs.

In this chapter, we review previous relevant work associated with relational models of incomplete information. As the existing research landscape and the scope of the thesis are both very rich, it is impossible to include all related work. Hence, we will focus on the most influential work. More specifically, Section 3.1 will review previous work on the interpretations of missing information in extensions of the relational model of data. Section 3.2 will review extensions of the notions of classical keys and functional dependencies. Section 3.3 will review previous work on Armstrong relations. Section 3.4
will review previous work on the discovery of keys and functional dependencies from data. Section 3.5 will review achievements on database schema design related to keys and functional dependencies. The main point of distinction from previous work is that we follow an approach that is independent of interpretations of missing data, while previous work has focused on specific interpretations. Finally, Section 3.6 will briefly summarize this chapter.

3.1 Relational Model of Incomplete Data

3.1.1 Relational Model

A database management system (DBMS) manages data in one or more persistent storages so that users can extract or update data efficiently in terms of time and space. Codd [27] introduced a relational model of data for database management systems, which later became the foundation of modern relational databases. Codd’s relational model is widely implemented in modern commercial DBMS such as Oracle Database and Microsoft SQL server.

There are two important advantages to relational models for data management applications. Firstly, relational models separate logical representations of data from their physical implementations. It facilitates the study of query languages [24], query evaluation [45], query optimization [25] and schema design [26]. Secondly, relational models have a strong semantic underpinning on modeling the real world. For example, relational models can identify real world objects as entities and allow cross-references among these objects. Semantic models of relations such as Codd’s integrity constraints [27] are important for building highly efficient databases, integrating data, and ensuring data quality.

One of the emerging demands requires managing missing information in DBMS. For instance, the DBMS in [98] is required to store fuzzy readings
Relational Model of Incomplete Data

from movement detection sensors. Allowing relational models to accommodate missing information can greatly enhance the power of existing applications such as Online Transaction Preprocessing (OLTP) and data warehouses.

3.1.2 Interpretations of Null Markers

Early work on the relational model heavily focused on complete data. For research to become more relevant to practice, incomplete data must be studied in details. In the relational model and its implementation, different kinds of incomplete information are uniformly represented by the occurrence of a special symbol, also called null marker. Different research has associated different meanings with the occurrences of these null markers. There are three main interpretations, as represented in the state-of-the-art research literature [112].

- **Unknown**: Some values exist but are currently unknown in a database and it could be updated later by database users;
- **Non-existent**: Some values do not exist at all e.g. inapplicable information;
- **No information**: A missing value only represents a placeholder for “unknown” or “non-existent” null makers.

The “unknown” interpretation was firstly discussed by Codd in [29]. In the paper, he uses a three-valued logic to interpret the result of an expression containing “unknown” null markers. That is, an expression is not only evaluated to be true or false but also to be “maybe”. Codd’s null substitution principle indicates that an expression is evaluated to “maybe” if and only if there exist non-null substitutions for all null markers in the expression such that it becomes true. Later on, Codd extended his model to adapt more complex situations when to deal with "unknown" null markers [30]. In the research, he gave a strong semantic underpinning on why his approach should be adopted. In addition, he showed how to deduce equalities between his
proposed null markers with *three-valued logic* in a relational database query. Although Zaniolo [112] explicitly stated that the “no information” interpretation is different from Lien’s “non-existent” interpretation [71], similar results have been established in the area of data dependencies [66].

### 3.2 Data Dependencies with Incompleteness

Data dependency is one of the most important semantic models used in relational databases. It investigates interrelationships between different tuples of the same schema. Research on data dependency not only enables relational models to capture meanings of the real world but also develops applications for *data cleaning* [93] and *database design* [13]. While about 100 different classes of data dependencies are known [102], keys and functional dependencies arguably constitute two of the most important classes.

#### 3.2.1 Keys

Keys have been studied in-depth on complete data [6], [69] and play a crucial role in database design [100]. To study keys of incomplete relations, many researchers have proposed different generalizations of keys besides functional dependencies. The notion of *key family* is discussed in [68], [73]. A key family over a relation schema is a family of sets of attributes. The satisfaction of a key family for an incomplete relation requires:

- The incomplete relation has to follow a possible world semantic i.e. null markers can be replaced by non-null values or inapplicable nulls and the possible worlds include all possible substitutions [30];

- Every possible relation is complete and satisfies the key which is the union of all sets in the key family;

- Tuples in the incomplete relation can be distinguished by some set of attributes in the key family.
In the early work of Codd [29], he stated two important principles of integrity constraints:

- (Entity integrity) Values of a primary key cannot be missing;
- (Referential integrity) Values of a foreign key must refer to the values of its corresponding primary key.

However, Levene and Loizou [68] assert that Codd’s principle of entity integrity is too strict. Many situations that incomplete tuples can be identified should be considered. Indeed, a key family captures the situation that a key is satisfied in all possible worlds of an incomplete relation and a key can be satisfied sometimes when some information is not applicable.

The study in [48] has proposed a new approach for axiomatizing keys of incomplete relations, which follows Codd’s principle of integrity constraint. Particularly, the study proposed a class of keys for incomplete relations that:

- tuples in a relation should be \( K \)-total if \( K \) is a key for the relation, and
- values of the attributes from a key should uniquely identify each tuple in a given relation.

For a complete relational model, two rules are required to form a sound and complete axiomatization:

- \( R\)-axiom: the set of all attributes over relation schema \( R \) constitutes a key if there is no other key;
- Superkey: any superset of a superkey is a superkey.

However, in the result of [48], \( R\)-axiom and superkey rule are no longer sound. Instead, a new axiom is established:

- If \( X \) and \( Y \) are keys of a relation, then the union of \( X \) and \( Z \) is also a superkey for the relation where \( Z \subseteq Y \).

The new notion not only shows a sound and complete axiomatization but also a solution to the underlying implication problem. Solving the implication problem is only in time linear with respect to the input keys.
Another generalization of keys is discussed in [59]. Similarly, the authors adopt a possible world semantic and allow inapplicable null markers in incomplete relations. Unlike key families, the authors follow the traditional notion of keys and regard the relation in a possible world as a multiset i.e. tuples with missing information are not necessary to be distinguishable in a SQL table. So, possible and certain keys are introduced to capture the situation that some tuples may appear to be identical but in fact different in some possible world. A certain key is satisfied by relations in all possible worlds of an incomplete relation. A possible key only has to be satisfied by some possible world of an incomplete relation. In addition, the study showed a sound and complete axiomatization for possible and certain keys. The proposed axiomatization also reveals interactions between possible and certain keys i.e. a possible key can be directly derived from a certain key and a certain key can be derived from possible keys in a null-free subschema. Most importantly, the implication problem for the class of possible and certain keys is solved in linear time in the size of the input.

3.2.2 Functional Dependencies

Functional dependency was introduced in [28]. It captures the situation that values of attributes can functionally or uniquely determine values of some other attributes. A sound and complete axiomatization system for functional dependencies was studied in [9]. Note that the axiomatization problem of keys is usually considered as the special case of a more general data dependency. The implication problem of functional dependencies was solved in [9] as well. Beeri and Bernstein [12] further improved the original algorithm to solve the same implication problem. Solutions to the implication problem of functional dependencies are extremely helpful in deciding equivalence between two sets of constraints, eliminating redundant keys and finding the non-redundant cover of a given set of constraints [6].
3.3 Armstrong Relations

Functional dependencies of incomplete relational model also have been studied extensively. Lien introduced functional dependency with nulls (NFD) and established a sound and complete axiomatization for NFD in [71]. The associated implication problem of NFD was discussed in [11]. The syntax of an NFD is similar to traditional functional dependency. Following the Codd’s principles of integrity constraints, an NFD $X \rightarrow Y$ only applies to $X$-total tuples i.e. the tuples that do not feature null markers in $X$. As a result, Lien found that the transitivity axiom is no longer sound for NFDs. When Atzeni and Morfuni [11] studied the interactions between NFD and existence constraints of nulls [75], in order to prove a sound and complete axiomatization, they introduced a new class of functional dependencies with nulls called fictitious functional dependency (FFD). There is a sound set of axiomatizations for FFD, which coincide with the axiomatizations in [9]. Unfortunately, there is no further study that aimed at discovering a complete set of rules for FFDs. In [66], authors generalized Armstrong’s and Lien’s axiom system by introducing strong functional dependency (SFD) and weak functional dependency (WFD). In order to coordinate two axiomatizations, the generalization uses SFD and WFD to express the possible worlds semantics i.e. given an incomplete relation $r$, $r$ satisfies an SFD $\delta$ if and only if all possible complete relations of $r$ satisfy $\delta$; $r$ satisfies a WFD $\delta$ if and only if there exists some complete relation of $r$ that satisfies $\delta$. The axioms for strong and weak functional dependencies are proven sound and complete. In the end, the authors also solved the implication problem of SFDs and WFDs with a polynomial time algorithm.

3.3 Armstrong Relations

An Armstrong relation is a user-friendly representation of data dependencies that shows examples of satisfied and unsatisfied constraints. Given a set
of data dependencies $\Sigma$ of class $C$, an Armstrong relation of $\Sigma$ satisfies a $C$-dependency $\sigma$ if and only if $\sigma$ is implied by $\Sigma$. Note that we use the terminology *Armstrong sample* sometimes when an Armstrong relation is constructed from real world data.

There is a broad class of data dependencies which enjoys Armstrong relations [39] i.e. there always exists an Armstrong relation given any set of functional dependencies [9]. Many researchers [17], [82] argued that Armstrong relations are helpful in identifying undesirable and acquiring meaningful data dependencies. A research has shown with an empirical evidence that Armstrong relations are not only useful by its structural and algorithmic properties but also support the acquisition of data semantics and the refinement of data constraints [65]. In the paper, the authors primarily investigated the usefulness of Armstrong relations of functional dependencies. An experiment has been demonstrated in details that teams of database designers tried to identify a set of desirable functional dependencies in an application domain. The experiment was conducted with the help of 20 different database design teams. At the beginning, 20 teams were asked to identify the set of functional dependencies that they perceived meaningful for a given application domain. Afterwards, the design teams were asked to revise their original sets by using Armstrong relations. In the end, the authors proposed two measurements in order to compare how meaningful the sets of functional dependencies that were originated by simply relying on the domain experts and by the assistance of Armstrong relations. Briefly, the proposed measurements are:

- **Soundness**: Given a target set of functional dependencies, how many of the user-identified functional dependencies are implied by the target set?

- **Completeness**: Given a target set of functional dependencies, how many of these are implied by the given user-defined set of functional dependencies?
The results concluded that Armstrong relations are not helpful in identifying meaningless functional dependencies which are mistakenly considered as meaningful but they are helpful in identifying actually meaningful functional dependencies which are mistakenly identified as meaningless by database designers.

In [66], the authors proved the existence of Armstrong relations in the axiomatization system of strong and weak functional dependencies. However, there is no more research on the related computation problem. In fact, the results in [59] showed that Armstrong relations for strong and weak functional dependencies do not exist. In [48], a new class of keys is introduced for relational models with nulls. The research showed the existence of Armstrong relations over their axiomatization system of keys. In addition, their results also solved the associated computation problem by an algorithm utilizing the notion of anti-keys. However, the computation complexity of the algorithm is precisely exponential i.e. the algorithm takes exponential time to obtain an Armstrong relation given a set of keys, and there exists a set of keys whose minimal Armstrong relations contain exponentially many tuples with respect to the number of attributes in the given set of keys. While studying uncertainties of relational databases, foundations are laid out for probabilistic keys and its Armstrong relations in [58]. The structural and algorithmic characterizations of Armstrong SQL tables for possible and certain keys are established in [59]. Similar to [48], the concept of anti-key plays an important role in constructing Armstrong SQL tables. Informative Armstrong databases are studied in [84]. The study inspires the idea that an Armstrong relation should be generated or sample from real world data. However, the study only focus on complete data and the related experiments are very limited.
Chapter 3. Literature Review

3.4 Data Dependency Discovery

The discovery of data dependency $C$ mines a cover of $C$ from a given relation. Next, we review the historical developments on discovering keys, unique constraints and functional dependencies, which are most relevant to this thesis.

3.4.1 Key and Unique Constraint

First of all, we want to stress that there is a strict distinction between unique column combinations (UCCs) and SQL’s unique constraints as far as we can tell from the reviewed literature. The difference is that a UCC evaluates $\bot = \bot$ to true, while an SQL unique evaluates it to false.

Giannella and Wyss introduced four column-based algorithms for the discovery of minimal keys [43]. The algorithms examine keys from an attribute lattice in a bottom-up, top-down or hybrid manner. The bottom-up approach generates a new key candidate if all its subsets from one lower level are not satisfied. The top-down approach generates a new key candidate whenever some of its superset from on higher level is satisfied. A simple hybrid approach uses the bottom-up and top-down approaches to ‘halve’ the search space. The hybrid approach makes the bottom-up and top-down approaches communicate with one another, so the attribute lattice is pruned faster. The experimental results are only restricted to synthetic datasets. Their algorithms do not scale well with large column numbers even with the hybrid approach.

Sismanis et al. introduced the GORDIAN algorithm to discover minimal keys [99]. A data structure called prefix tree was introduced to efficiently extract column combinations known as non-keys, which are considered as invalid keys in the input relation. The algorithm performs well on datasets with large numbers of rows and columns in some cases. The experimental
results on real world datasets are limited, and the research does not discuss how to handle missing values.

Abedjan and Naumann introduced the Histogram-Count-based Apriori Algorithm (HCA) in [3] to discover UCCs. The algorithm efficiently generates UCCs with different cardinalities from small to large. By counting the frequencies of distinct values, the algorithm can validate whether a UCC is satisfied by the given relation. The HCA algorithm also employs the prefix tree from the GORDIAN algorithm, such that non-keys can be used to prune UCCs without checking value histograms. The study does not handle missing values and the experiments are only conduct on the real world datasets where only a small number of UCCs exhibit.

Heise et al. introduced a scalable UCC discovery algorithm in [50]. The algorithm models the search space as an attribute lattice but implements two search strategies that are different from the usual level-wise search. Starting from a search point (a UCC), a greedy strategy keeps searching for new UCCs which are most likely to be satisfiable if the given UCC is not already satisfied; and a random-walk strategy keeps searching for supersets (subsets) of the starting search point and randomly switches to a new search point when the current UCC is satisfied (unsatisfied). However, only the random-walk strategy can be parallelized. Missing values are considered but do not conform to SQL’s unique constraint. Figure 10 in [50] shows that the algorithms scale poorly on a large number of attributes if an SQL’s unique semantics is adopted.

Papenbrock and Naumann introduced a hybrid algorithm for the discovery of UCCs in [90], which is based on their hybridization strategies for functional dependencies (FDs) [89]. The hybrid algorithm switches back and forth between a column-based and row-based algorithm. Briefly, the column-based algorithm validates UCCs in an attribute lattice in a bottom-up manner and switches to the row-based algorithm when too many invalid UCCs are found. The row-based algorithm extracts counter-examples heuristically
from a given relation and switches to the column-based algorithm whenever too few counter-examples are found.

3.4.2 Functional Dependency

TANE [52] is one of the earliest algorithms developed for functional dependency discovery. Briefly, the algorithm builds an attributes lattice where elements are sets of attributes and a partial order that is the inclusion between sets. Then, it traverses the entire lattice to find which attribute the current set of attributes imply in a given relation. To improve the running time of TANE, the authors proposed a level-wise pruning strategy which checks attribute sets iteratively and stops if it finds the smallest set of attributes that implies a specified attribute i.e. at each iteration, all sets of attributes have the same cardinality and the cardinalities of the sets of attributes will only increase by one for the next iteration. So, the lattice becomes smaller if more sets are found before. Following this direction, algorithms such as FUN [86], FD_MINE [111] and DFD [4] were introduced but with different pruning and lattice traversal strategies. Note that several important techniques such as stripped partitions and prefix blocks were introduced in [52], and greatly improve the run-time efficiency of column-based discovery algorithms.

FastFD [110] computes functional dependencies from difference sets. A difference set of a pair of tuples is a set of attributes where the tuples differ in their values. Intuitively, a difference set captures possible LHS attributes for some RHS attributes. Then the problem is immediately reduced to computing the minimal transversal of given difference sets. FastFD implements an optimized and non-repeated strategy to compute the minimal transversal. It has been shown in the experiments that FastFD’s running time is much better than a brute-force algorithm even though both algorithms have the same computation complexity in the worst case. FastFD is an improvement to Dep-Miner algorithm [74] which uses agree sets to find the negative cover of functional dependencies in a given relation.
Flach and Savnik introduced a more intuitive row-based algorithm called FDEP in [41]. Although agree sets are computed beforehand, the algorithm only uses them as necessary. In fact, agree sets are viewed as counterexamples from a given relation i.e. an FD must not be satisfied by the given relation. Then, these counterexamples can be iteratively applied to some candidate FDs i.e. FDs are assumed to be satisfied. If some candidate FD contradicts to a counterexample, then it can be augmented accordingly. They also introduce an efficient data structure called *FD-tree* to quickly search candidate FDs that contradict to some counterexample.

The studies in [87] survey the aforementioned algorithms. The purpose is to test the performances of these algorithms on a family of benchmark datasets. The time complexity of the tested algorithms in [87] are dependent either on the number of attributes or rows in a dataset. However, these algorithms only work on complete relations but some of the benchmark datasets are incomplete i.e. some tuples contain null markers. Since the research only aims to compare performances between these algorithms, for simplicity, all the null markers in the experiments are treated as equals. However, real world applications should be more precocious with null markers. Over-interpreting null markers may lead to concluding incorrect application insights. In data cleaning, one of the main tasks is to check if the collected data conforms to the integrity constraints in a data warehouse. So, mis-interpreting null markers can cause a loss of information or updating overhead to the data warehouse. Particularly, if a data cleaning process treats all null makers as different, then it can potentially cause more violations to the specified integrity constraints. In other words, less information will be integrated into a data warehouse. Conversely, if a data cleaning process treats all null markers as equal, it may cause heavy overheads for deleting information in a data warehouse if some null markers can be updated later.

Lastly, Papenbrock and Naumann introduced a hybrid algorithm to discover *FDs* in [88], which is similar to [90]. The hybrid algorithm switches
back and forth between the column-based and row-based algorithm. Briefly, the column-based algorithm validates functional dependencies of a FD-tree from lower to higher levels and then switches to the row-based algorithm when too many functional dependencies are invalidated. The row-based algorithm extracts counterexamples heuristically from a given relation and switches to the column-based algorithm whenever too few counterexamples could be found.

3.5 Normalization

*Boyce-Codd Normal Form* (BCNF) and *Third Normal Form* (3NF) are important tools for database schema design. On one hand, any relation schema enjoys lossless BCNF decompositions that eliminate all data redundancies caused by FDs, but may not be dependency-preserving. On the other hand, every relation schema enjoys lossless 3NF syntheses that are guaranteed to be dependency-preserving, but may not eliminate all data redundancy. A recent research shows how to combine classical FD discovery with classical BCNF-decomposition to drive schema normalization from data [91]. However, the designs generated in the research lacked data quality criteria, and handled null markers like any other value.

Schema design for data with missing values has been a long-standing open problem [47], [60], [69]. Almost exclusively, the main focus of the research has been on suitable extensions of FDs to incorporate null markers. In that area there is a plethora of research, mostly focusing on foundational aspects such as reasoning. Among those extensions, there are some approaches where null-free sub-schemata have been considered for reasoning about FDs [11], [49]. That work mainly focused on reasoning rather than schema design. In particular, the approach is restricted because null-free sub-schemata do not permit any null marker occurrences in the columns of the sub-schema. Weak and strong FDs extend classical FDs by interpreting missing values
3.6 Summary

In the context of relational models that encompass incomplete data, many extensions of keys and functional dependencies have been studied. However, all of the extensions are founded on specific interpretations of null marker occurrences. Because of this, some of the extended notions such as Codd keys [48] and NFDs [11] do behave quite different from the well-understood concepts of classical keys and functional dependencies, and have therefore

with possible worlds semantics [66], [69]. As with other extensions, the semantics depends strongly on the interpretation of the null marker occurrences. This makes it difficult to address modern applications, such as data integration, where different null marker occurrences may require different interpretations. A second limitation is that the complexity of reasoning becomes often prohibitively expensive to guarantee efficient schema designs [69]. The concept of possible and certain FDs was introduced to show a suitable extension of schema design for data with missing values, at least in terms of BCNF [60]. The work contains a review and comparison of FD extensions to data with missing values. In summary, these approaches focus on the interpretation of null markers, aiming at their inclusion in decisions about schema design. These approaches can be criticized in different respects. Firstly, it is doubtful whether missing information should have an impact on schema design decisions. Secondly, modern applications such as data integration accommodate missing data values that require different interpretations, which makes it difficult to justify these approaches. Thirdly, the approaches have not led to generally good solutions in the past. Finally, application requirements have not been considered in these approaches, even though design decisions should be based on them.
higher barriers of use by practitioners. On the other hand, some of the extended notions such as key families [68] and weak/strong functional dependencies [66] can only be used for reasoning tasks but have not been shown to enjoy capabilities for other applications such as discovery and schema design. Although recent developments on possible/certain keys [59] and possible/certain functional dependencies [60] have shown promising results on a range of applications, including reasoning tasks and schema design, there is still a lack of strong evidence that these new notions can be adopted by real-world applications and practice.

Besides, Armstrong relations and the output of dependency discovery algorithms provide helpful and comprehensive meta-information, which can dramatically facilitate the benefits of data profiling, data cleaning, and schema design from a data-driven perspective. Surprisingly, little attention has been devoted towards the notions of keys and functional dependencies that encompass null marker semantics. As a consequence, the use cases of previous research remain unclear. For example, most of the functional dependency discovery algorithms simply treat null marker occurrences like any domain value occurrence. In practice, such situations require a substantial amount of manual labor on inferring, cleaning and justifying missing values, before any advanced tools and analytics can be applied to actual data. What is potentially worse is that the outputs of dependency discovery remain inapplicable because the semantics of the discovered dependencies does not comply with the semantics required by a given application. One future approach is to clarify which applications apply which semantics, and then tailor the dependency discovery algorithms to the semantics that is required. The approach of this thesis is to step away from different interpretations and simply rely on the complete fragments of the data.

Compared to previous work, embedded unique constraints (eUCs) and embedded functional dependencies (eFDs) are based on a semantics that is robust
under different interpretations of missing data. In addition, users can declare application requirements on data completeness and data integrity as part of these constraints, and therefore these data quality dimensions to a first-class citizen status for their applications. Indeed, eUCs and eFDs are able to enforce not only integrity but also completeness requirements on any of the given datasets. As we will show in this thesis, it is possible to establish efficient solutions to a wide range of important computational problems, including reasoning, sampling, discovery, and schema design using eUCs and eFDs.
Chapter 4

Ranked Syntactic and Semantic Data Profiles for Functional Dependencies

The discovery problem of functional dependencies has been extensively studied. Despite the likely intractability, a recent state-of-the-art algorithm shows strong evidence that the discovery problem of FDs can be solved efficiently for many real-world data of relational model [90]. In this chapter, we introduce a new hybrid algorithm that is enabled to take advantage of tradeoffs between runtime efficiency and memory consumption. Using extensive experiments we establish compelling evidence that our proposed algorithm advances state-of-the-art on FD discovery in terms of runtime efficiency, column-, and row-scalability. In addition, we establish three novel main contributions on the output of discovery algorithms. Firstly, previous work has focused on so-called left-hand side reduced covers of the FDs that hold on a given relation. Such representation still suffers from a lot of redundancy, in the sense that some of the FDs are implied by other FDs in the LHS-reduced cover. We will show that canonical covers can greatly reduce the output size, and can usually be computed without significant overheads. Secondly, previous work simply lists the discovered dependencies, without distinguishing between their relevance for the given dataset. Our novel contribution is to rank the discovered FDs by the number of redundant data
value occurrences they cause in the dataset. Hence, users of these tools are
guided towards dependencies in the output that are more relevant. Listing
the FDs that hold on a given dataset can be understood as a syntactic data
profile, because it simply enumerates syntactic statements. Our third contri-
bution towards the outputs of FD discovery is to utilize semantic profiles in
the form of Armstrong samples. These are data samples of the given dataset
which satisfy the same set of FDs. Such semantic data profilers draw the at-
tention of users to the actual semantics of the discovered FDs, in particular
as the size of the Armstrong samples is typically much smaller than that of
the original dataset.

We will provide a motivation for our work and an overview of our results
in Section 4.1. Section 4.2 revisits fundamental techniques of previous algo-
rithms and comments on avenues of improvement. Section 4.3 then develops
our new algorithm and illustrates the techniques on examples. Section 4.4
contains the analysis of our experimental results. The outcomes of our ex-
periments with canonical covers are presented and discussed in Section 4.5.
Section 4.6 describes our proposal for ranking FDs, and an analysis of our ex-
periments with this notion. Section 4.7 then describes our results around the
experiments with semantic data profiles in the form of Armstrong samples.
Section 4.8 contains a brief summary about this chapter.

Some of the results of this chapter have been reported in the article “Dis-
covery and Ranking of Functional Dependencies”, which has been accepted
by the 35th IEEE International Conference on Data Engineering (ICDE2019)
[107].

4.1 Introduction

Data profiling comprises the activities that determine meta data about a
given dataset [1]. In practice, data profiling is a scientific approach towards
data preparation, a resource-intense task in data science projects. Applications of data profiling include data integration, data quality, data cleaning, data preparation for analytics, query optimization, and data repository design [1]. A fundamental task in data profiling is the discovery of data dependencies that hold on the given dataset. Since the 1980s many advances have been made. We will focus on functional dependencies (FDs) in this chapter. These have received most attention from academia and industry, due to their usefulness in many applications [4], [22], [41], [52], [74], [80], [86], [88], [110], [111].

The discovery problem for FDs is to compute the set of FDs that are satisfied by a given relation. This problem is computationally challenging. There are relations over any given number of columns (attributes) whose best representation of the satisfied FDs is of exponential size [81]. The decision variant is to decide for a given relation $r$ and a given positive integer $k$ whether there is an FD $X \rightarrow A$ with $A \notin X$ and $|X| \leq k$ that is satisfied by $r$. The decision variant is NP-complete [32] and even W[2]-complete [20]. Despite these fundamental barriers on generally efficient solutions, known algorithms can efficiently solve many real-world instances of the discovery problem. For example, the class of row/column-efficient algorithms provide efficient solutions whenever the given dataset has few columns/rows, respectively. However, real-world data, especially big data, have typically many rows and columns. The recent hybrid algorithm [88] combines row- and column-based approaches to address larger datasets. For example, it can find the 40,195 LHS-reduced FDs (Definition 2.16) that the benchmark dataset *diabetic* with 101,766 rows and 30 columns exhibits in 2,865 seconds using 2,253 MB of main memory. In the hybrid algorithm, a switch of strategies occurs whenever the current strategy is not working well, that is, if either too many FDs are invalidated or too few invalid FDs are found. However, a switch from the current strategy is never based on evidence that the other strategy will be successful. The lack of evidence leaves room to improve efficiency and
Chapter 4. Ranked Syntactic and Semantic Data Profiles for Functional Dependencies

T able 4.1: Syntactic and semantic profiles for pdb with over 17 million rows and 13 columns

<table>
<thead>
<tr>
<th>Informativ e Armstrong sample (Example of semantic profile)</th>
<th>Canonical cover (Example of syntactic profile)</th>
</tr>
</thead>
<tbody>
<tr>
<td>199768 321 B 1 3 ILE B 3 3 3 ILE ILE .</td>
<td>{0→1, 6, 7}</td>
</tr>
<tr>
<td>194400 322 C 3 489 ARG A 489 0 ⊥ ⊥ ⊥ .</td>
<td>{1, 2→6}</td>
</tr>
<tr>
<td>176755 279 B 1 3 ILE B 3 3 3 ILE ILE .</td>
<td>{1, 3, 5, 9→10}</td>
</tr>
<tr>
<td>176752 279 A 1 189 SER A 189 189 ⊥ ⊥ ⊥ .</td>
<td>{1, 3, 7→5}</td>
</tr>
<tr>
<td>194399 322 C 3 489 ARG A 489 0 ⊥ ⊥ ⊥ .</td>
<td>{1, 5, 9, 11→10}</td>
</tr>
<tr>
<td>194398 322 C 3 487 ALA A 487 0 ⊥ ⊥ ⊥ .</td>
<td>{1, 6, 2→9}</td>
</tr>
<tr>
<td>176751 279 A 1 188 SER A 188 188 ⊥ ⊥ ⊥ .</td>
<td>{1, 6, 7→8, 11}</td>
</tr>
<tr>
<td>194397 322 C 3 486 ASP A 486 0 ⊥ ⊥ ⊥ .</td>
<td>{1, 6, 8, 10→11}</td>
</tr>
<tr>
<td>195367 320 D 1 191 LEU D 191 191 191 LEU LEU .</td>
<td>{1, 6, 9, 11→9}</td>
</tr>
<tr>
<td>195447 320 C 1 191 LEU C 191 191 191 LEU LEU .</td>
<td>{4→7}</td>
</tr>
<tr>
<td>195124 326 C 3 133 GLN A 133 307 307 GLN GLN .</td>
<td>{5, 7, 9, 11→10}</td>
</tr>
<tr>
<td>195121 326 C 3 130 GLN A 130 304 304 GLN GLN .</td>
<td>{7→4}</td>
</tr>
<tr>
<td>176749 279 A 1 186 ASP A 186 186 ⊥ ⊥ ⊥ .</td>
<td>{8→12}</td>
</tr>
<tr>
<td>176748 279 A 1 185 THR A 185 185 ⊥ ⊥ ⊥ .</td>
<td></td>
</tr>
</tbody>
</table>

scalability, which can make datasets with more rows, columns, and FDs accessible to FD discovery. Our major contribution is a new hybrid algorithm with innovations in strategy and technology. Strategically, we switch from a column- to a row-based approach whenever it is likely that many FDs can be validated. Technically, this is made possible by a novel data structure and the first algorithm that computes stripped partitions dynamically. Experiments show that our algorithm leverages conservative main memory resources to outperform the state-of-the-art in run times, row- and column-scalability. For example, it discovers the 40, 195 LHS-reduced FDs of diabetic within 848 seconds using 4,301 MB of main memory.

The aim of discovery algorithms is to represent all the valid FDs of a given dataset efficiently. In previous work the representation is a LHS-reduced cover, minimizing the LHS \( X \) of FDs \( X \rightarrow Y \) i.e. Definition 2.16. We further deliver three novel insights for discovered FDs to overcome their shortcomings in representation.
Firstly, LHS-reduced covers may contain many redundant FDs. Non-redundant representations are smaller, and easier to process for computers and humans. Our experiments show how quickly canonical covers can be computed and how much they reduce output sizes. They achieve an average of 50% savings on benchmark data. Ten of our datasets are smaller and achieve a reduction by 25%, while the remaining datasets are larger and achieve a reduction by over 70%. For example, a canonical cover for \textit{ncvoter1k} consists of only 185 FDs with 927 total attribute occurrences, reducing the LHS-reduced cover by approximately 4 times in size. The canonical cover can be computed in 0.023 seconds from the LHS-reduced cover.

Secondly, the FDs that discovery algorithms produce are syntactic profiles. These are targeted towards people that understand what FDs are, such as business analysts. Most users do not understand what FDs are. This means that current discovery algorithms do not produce user-friendly profiles. Here, we promote the use of semantic profiles in addition to syntactic profiles. Semantic profiles are defined as informative Armstrong samples, that is, samples of the given dataset that satisfy the same FDs [34]. Semantic samples also offer a complementary computational approach. A powerful example is the dataset \textit{uniprot1k}, one of the benchmark datasets for discovery algorithms. So far, no algorithm is known computes the result for this dataset, consisting of 1,000 rows and 223 columns. The dataset satisfies too many FDs, and even high memory machines do not help. Interestingly, we can compute a semantic profile of this dataset. This is not by accident. Syntactic profiles compute the valid FDs, while semantic profiles compute the invalid FDs. These numbers can be different, resulting in computational advantages for the approach with smaller numbers. Another illustration of our ideas is given by the protein dataset \textit{pdb}. It has 13 columns and over 17 million rows. State-of-the-art algorithms produce the LHS-reduced cover in Table 4.1, with 68 FDs. A canonical cover (see Definition 2.17), as shown on the right of Table 4.1, has only 14 FDs. Our semantic profile has only 14 rows but satisfies...
the same FDs as the original dataset. We computed the sample in just over 220 seconds.

Thirdly, the output of FD discovery algorithms is not ranked. The more FDs are returned for a given dataset, the more difficult it becomes for users to assess their relevance. As stated before [90], ultimately, a domain expert must assess whether an FD is meaningful for the application domain. Even though FDs that only hold accidentally on the dataset are still useful for some applications, such as query optimization, it is still beneficial to automatically rank the relevance (see Definition 4.2) of the discovered FDs for the given dataset. We propose data redundancy as a natural measure of relevance. It is natural for at least two reasons. 1) FDs are major source for data redundancy, having brought forward Boyce-Codd and Third Normal Form proposals [77], [104]. Consequently, the number of redundant data values caused by an FD indicates the relevance of this FD for normalization. 2) Data redundancy caused by an FD $X \rightarrow Y$ measures how many instances of the pattern “$X$-value determines $Y$-value” actually occur in the dataset, again showing how relevant the pattern is. We will show a more detailed qualitative analysis of FD relevance in Section 4.6.2 of this chapter.

We summarize our contributions in this chapter as follows.

1. We propose a new hybrid FD discovery algorithm based on innovations in strategy and technology. Strategically, we switch from a sampling-based discovery to a validation-based discovery whenever it is likely many FDs can be validated. Technically, this is made possible by a novel data structure and the first algorithm that computes stripped partitions dynamically. Extensive experiments demonstrate that our algorithm outperforms the state-of-the-art in discovery times, row- and column-scalability.

2. We demonstrate that the output size of FD discovery algorithms can be reduced significantly by canonical covers. These computations are fast and achieve an average 50% savings in time on benchmark data. Ten
of our datasets are smaller and achieve a run time reduction by 25%, while the remaining eleven datasets are larger and achieve a run time reduction by over 70%.

3. We propose direct and indirect approaches to computing semantic profiles.

4. We propose to rank the outputs of FD discovery algorithms by the number of redundant data value occurrences they cause in the given dataset. Applying this ranking to the FDs exhibited by real-world benchmark data, a quantitative and qualitative analysis illustrates that our measure can provide effective guidance for data stewards in assessing the relevance of discovered FDs.

5. We report results for the two most common interpretations of missing values. Since values are missing frequently, such distinction is important for applications.

4.2 Revisiting Previous Algorithms

Before introducing the dynamic hybrid algorithm (DHyFD), we first revisit previous column-efficient [41] and row-efficient [52] approaches as well as the state-of-the-art hybrid algorithm [88] to comment on opportunities to improve them.

4.2.1 Row-efficient Algorithm

The column-based algorithm in [52] models the search space of FDs as an attribute lattice. The algorithm traverses the lattice from bottom to top. At each level, the LHS and RHS of an FD are attribute sets. Each pair of LHS and RHS is validated using the stripped partition of the LHS. If an FD does not hold, new LHSs on the next level for the same RHS attribute are generated. The stripped partition of the new LHSs are computed. This process is infeasible
if there are too many columns. Generating LHSs by levels typically enumerates the entire lattice if valid FDs exist at different levels. Stripped partitions duplicate the original input aggressively, consuming any available memory for inputs with too many rows.

Stripped partitions are efficient for FD validation. Validating $X \rightarrow Y$ usually requires a mapping from the $X$-values to their $Y$-values. Once duplicated $X$-values are found, unmatched $A \in Y$-values invalidate $X \rightarrow A$. This method is inefficient on larger inputs. For example, validating $X \rightarrow A$ and $XB \rightarrow A$ (assume $X \rightarrow A$ is invalid) creates mappings for $X$- and $XB$-values. Hence, many $X$-values are computed redundantly. Such redundancy causes inefficiency if the input contains too many rows and columns. If $\pi_{XB}$ could be generated dynamically from $\pi_X$, then only $B$-values needed extraction. Computation would become more efficient. Hence, using previously computed stripped partitions can decrease the cost of FD validation effectively. Another challenge is to let stripped partitions consume only reasonable memory.

### 4.2.2 Column-efficient Algorithm

The column-efficient algorithm of [41] uses agree sets of two tuples. These consist of those attributes on which the two tuples have matching values. That is, $agr(t,t') = \{A \in R \mid t(A) = t'(A)\}$. The agree set of $r$ is the set of agree sets for all pairs of distinct tuples in $r$, that is, $agr(r) = \{agr(t,t') \mid t,t' \in r, t \neq t'\}$. Importantly, the agree set for each pair of distinct tuples implies the non-FD $agr(t,t') \not\rightarrow R - agr(t,t')$ (see Definition 2.20). Starting with the FD $\emptyset \rightarrow R$, the column-efficient algorithm processes the agree sets in $agr(r)$ iteratively, inducing new FDs that do not contradict any of the non-FDs implied by the agree sets that have been processed so far. More details are discussed in Section 4.3.2.

Inducing FDs from non-FDs uses a tree-like data structure called FD-tree [41]. As shown in Figure 4.1, the LHS of an FD is represented by a path
4.2. Revisiting Previous Algorithms

in the FD-tree where each node represents an attribute in the LHS with labels of RHS attributes. An FD-tree provides quick access to all FDs that do not contradict any of the non-FDs processed so far. A new observation is that the induction algorithm of [41] only handles singleton RHS of implied non-FDs. Instead of inducing new FDs based on the non-FD $X \not\rightarrow R - X$, the induction process iteratively induces new FDs based on the invalid FDs $X \not\rightarrow A$ for all $A \in R - X$. FD-trees are implemented as a linked data structure. Path traversals are costly in practice since links are maintained by heap memory. Our improvement can thus reduce time spent on FD induction. These time are significant as the number of non-FDs (including duplications) can be quadratic in the given number of tuples.

4.2.3 Hybrid Algorithm

The sampling-focused hybrid algorithm [88] demonstrates that combining row- and column-efficient techniques can handle instances with more rows and columns. The strategy, illustrated on the left of Figure 4.2, has three components and two phases. On input $r$ over $R$, the sampling component computes a stripped partition for each attribute in $R$. Then the sorted neighbourhood pair selection method [51] extracts non-FDs by sampling agree sets of the stripped partitions. The validation component validates candidate FDs, and uses invalid FDs to update an FD-tree. The induction phase uses either the non-FDs from
the sampling component or the invalid FDs from the validation component to induce new FDs. The algorithm in [88] starts with the sampling phase, and then applies invalid FDs to update the FD-tree. Once too few new samples (non-FDs) are generated, the algorithm switches to the validation phase. Likewise, it switches back to the sampling phase if too many FDs are invalidated.

The hybrid algorithm implements the column-efficient algorithm as induction component. No effort is spent on reducing the time of traversing its FD-tree. The validation component only uses a fixed set of stripped partitions computed by the sampling component at the beginning. Hence, the algorithm does not take advantage of the row-efficient approach in which stripped partitions reduce redundant computations of values on the LHS of FDs. However, adapting the validation method from the row-efficient algorithm proves challenging. Here, a larger stripped partition is computed by joining the stripped partitions of two adjacent prefix blocks [52]. For example, $\pi_{XAB}$ is computed by intersecting $\pi_{XA}$ and $\pi_{XB}$. Hence, all invalid LHSs need to be known before the stripped partitions on a higher level can be computed. Due to the randomness in FD induction some of the invalid LHSs are eliminated by invalid FDs. This means each level of the FD-tree contains only some but not all of the invalid FDs. Hence, FD induction can make the computation of stripped partitions obsolete. The major challenge in hybridizing the column- and row-based algorithms is to dynamically compute stripped partitions while avoiding excessive memory consumption.

It is always better not to use invalid FDs from the validation component if more general non-FDs can be found. Suppose $X \not\rightarrow Y$ is an invalid FD over $R$. The invalid FD $X \not\rightarrow Y$ could be inefficient in two ways. Firstly, there may be a non-FD $X \not\rightarrow R - X$ such that $Y \subseteq R - X$. Secondly, the invalid FD $X \not\rightarrow Y$ may induce new FDs that could still be invalidated by a non-FD $X' \not\rightarrow R - X'$ where $X \subseteq X'$. 
4.3 Dynamic Hybrid Algorithm

Despite extensive previous research there are still opportunities for improvements. We introduce the dynamic hybrid algorithm for FD discoveries (DHyFD). The main strategy is briefly demonstrated on the right of Figure 4.2. Firstly, the new algorithm uses the sorted neighbourhood pair selection method to extract a wide range of non-FDs from the given relation. Secondly, the sampled non-FDs are used to update an extended FD-tree to derive new FDs with a synergized induction algorithm. Thirdly, we use dynamic data manager (DDM) to maximize sampling and validation efficiency. The purpose of DDM is to prepare suitable stripped partitions for FD validations. In particular, DDM refines stripped partitions for FD candidates efficiently by analysing trade-off between the use of memory by stripped partitions and the costs of validation. Lastly, after validating FDs, new non-FDs are extracted from the validation component, and applied to the extended FD-tree before continuing to the next level. In a nutshell, the efficiency of DHyFD requires extended FD-tree and DDM to work closely. On one hand, DDM provides efficient stripped partitions to validate FDs of the extended FD-tree. On the other hand, the FD-tree helps DDM evaluate validation efficiency to avoid excessive computation of stripped partitions.
Before going into details, we outline DHyFD as illustrated in Figure 4.3. Given a relation \( r \) over schema \( R \), DDM pre-computes stripped partitions for each singleton attribute of \( R \). The stripped partitions are refined later dynamically. Before any iteration, a set of initial non-FDs (aka initial samples) are sampled from the pre-computed stripped partitions. This step extracts a wide range of non-FDs to perform synergized induction on the extended FD-tree, providing a good first approximation of the final FD-tree. DHyFD then starts validating the extended FD-tree level by level. The current level is called the validation level \( (vl) \). DHyFD validates the FDs on the current \( vl \) by stripped partitions from DDM. During validation, a set of non-FDs is generated. Afterwards, synergized induction uses these non-FDs to derive new FDs. After induction, DHyFD decides if DDM can perform better by refining stripped partitions. The decision applies a novel measure called efficiency-inefficiency ratio. We refer to the validation level \( vl \) at which the latest refinement occurs as the controlled level \( (cl) \). In essence, the refined stripped partitions are based on the FDs associated with nodes at the controlled level. At the end of an iteration, DHyFD either computes the nodes at the next validation level or terminates if the extended FD-tree does not require further
traversal.

Next, we provide the details of each component.

### 4.3.1 Extended FD-tree

FD-trees were introduced in [41] as the underlying data structure to facilitate FD induction. We enhance FD induction by extended FD-trees which not only provide better support on deriving FDs but also facilitate efficient FD validations by communicating with DDM.

**Definition 4.1 (Extended FD-tree)** Let $\Sigma$ be a set of FDs over the relation schema $R$. Assuming attributes of $R$ are integers, an extended FD-tree has the following properties:

1. A unique root node represents the empty LHS;
2. Each node represents an attribute in $R$ except the root node;
3. Each node only has children of larger attributes;
4. For each FD $A_1 \ldots A_n \rightarrow Y \in \Sigma$, there is a path representing $A_1 \ldots A_n$ where $A_n$ is called an FD-node;
5. Each FD-node is associated with a non-empty attribute set representing the RHS of an FD;
6. Each node is assigned a positive integer $id$;
7. The default id of a node $A \in R$ is $A$;
8. Given a DDM of relation $r$ over $R$ and a node with id $i$ where $i > |R|$, the $(i - |R|)$-th stripped partition in the DDM is $\pi_{X'}$ where $X' \subseteq X$ and $X$ is the path from the root to the node.

**Example 4.1 (Extended FD-tree)** The right of Figure 4.1 shows an extended FD-tree of the FDs $A \rightarrow B$, $AB \rightarrow CD$, $CD \rightarrow B$. The integer ids are shown at the
bottom right. The extended FD-tree has less RHS labels than the FD-tree on the left of Figure 4.1.

The novelty of extended FD-trees is a new type of node, called FD-node, which stores RHS attributes of FDs. In contrast, nodes of an FD-tree store RHS attributes not only for the FDs represented by themselves but also for FDs represented by their descendants. In the left of Figure 4.1 the root node and the nodes at level 1 all have $B$ as a RHS attribute even though only node $A$ represents FD $A \rightarrow B$. This overhead in labeling is inefficient in practice. The FD-tree requires more effort on maintaining RHS attributes compared to extended FD-trees. Moreover, excessively tracking RHS attributes does not accelerate FD search. For example, on the left of Figure 4.1, if we search for $X \rightarrow B$, checking nodes that lead to FDs with attribute $B$ on the RHS does not prune the search space: indeed, the root node and the nodes at level 1 all have $B$ as their RHS attributes.

In DHyFD, FD-nodes are validated level by level. We call the level where the FD-nodes are validated the validation level. A DDM will also store an array of stripped partitions which are computed from the paths of length $l$ in the extended FD-tree. Here, the so-called controlled level $l$ must be smaller than the validation level. In fact, the integer ids of nodes in an extended FD-tree are assigned by DDM. They index the array of stripped partitions in the DDM. We say a node’s id is (in)consistent to a DDM if the attribute set $X$ of stripped partition $\pi_X$ is (not) a subset of the path that leads to the node.

In fact, the underlying FDs of FD-nodes in an extended FD-tree can be validated using the stripped partitions from DDM since the consistent id of a node refers to a general stripped partition i.e. if the difference between a controlled level and a validation level is greater than 0, the stripped partition $\pi_{X'}$ still can be used to validate the node’s underlying FD $X \rightarrow Y$ because $\pi_{X'}$ can be further refined to $\pi_X$ (see Algorithm 5 later for more details). The
4.3. Dynamic Hybrid Algorithm

intuition behind is that the difference between a controlled level and a validation level should be smaller if the current extended FD-tree potentially have many true FDs because validating true FDs is inevitable to compute stripped partitions which are specific to the FDs. Vice versa, the difference between a controlled level and a validation level should be bigger because invalidating a non-FD only requires to find a pair of tuples as a counterexample and it is unnecessary to compute the stripped partition of the LHS of a non-FD.

Hence, a major task of extended FD-tree is to make sure that only consistent ids are assigned to the nodes in a tree. During FD validations, DHyFD has to know all the nodes at the current validation level so that the nodes at the next level can be traversed. However, after validating current FD candidates, FD induction process may introduce new nodes by inserting a completely new path or extending an existing one (see Example 4.2). When a new path can extend an existing path, ids of the new nodes in the extended path should copy the consistent id in the existing path.

Example 4.2 (Update FD-tree) Let FD $AC \rightarrow E$ be the only path in an extended FD-tree over $R = \{A, B, C, D, E\}$. If non-FD $AC \not\rightarrow BDE$ is applied to the tree, $ABC \rightarrow E$ and $ACD \rightarrow E$ are induced. To add $ACD \rightarrow E$, RHS $E$ of FD-node C is removed and a child FD-node $D$ is appended to node $C$. In the end, node $C$ is no longer an FD-node and its child FD-node $D$ stores $E$ as RHS. To add $ABC \rightarrow E$, a new path $ABC$ is created from node $A$ since the only existing path is $ACD$.

Suppose the validation level is 2 before non-FD $AC \not\rightarrow BDE$ is implied. One can only retrieve FD-node $C$ from level 2 of the tree. After induction, node $B$ from the new path $ABC$ is added to level 2. Without knowing the new node $B$, it is impossible to validate $ABC \rightarrow E$ by only exploring the children of node $C$ at level 2.

Algorithm 1 assigns consistent ids while adding a new FD path. The algorithm checks if the inserted FD requires new nodes at the end of some path (step 5-8). If it does, the new nodes copy consistent ids from their ancestors at
the controlled level (step 11 - 14). The new nodes are added to the validation level (step 15) for complete traversal of the FD-tree.

Algorithm 1 Add FD

1: **Input:** An FD $X \rightarrow Y$ over $R$, the root node $root$ of an FD-tree, the controlled level $cl$, the set $vl\_nodes$ of all nodes at the validation level $vl$
2: **Output:** a new FD-tree with FD $X \rightarrow Y$
3: $current \leftarrow root$
4: $i \leftarrow 1, n \leftarrow |X|$
5: while $i \leq n$ do
6:   if $current$ has child $c$ of $A_i \in X$ then
7:     $current \leftarrow c, i \leftarrow i + 1$
8:   else break
9: while $i \leq n$ do
10:   Create a new node $c$ of $A_i \in X$ as the child of $current$
11:   if $i > cl$ then
12:     Assign id of $current$ as id of $c$
13:   else
14:     Assign the order of $A_i$ in $R$ as id of $current$
15:   if $i = vl$ then $vl\_nodes \leftarrow vl\_nodes \cup \{c\}$
16:   $current \leftarrow c, i \leftarrow i + 1$
17: Let $rhs(current)$ be the RHS of $current$
18: $rhs(current) \leftarrow rhs(current) \cup Y$
19: **Return** $root$

4.3.2 Synergized Induction

Given a non-FD $X \not\rightarrow Y$ over $R$, classical FD induction [41] updates a set of FDs over $R$ with respect to $X \not\rightarrow A$ for all $A \in Y$. In other words, the classical method suggests multiple traversals in an FD-tree for a non-FD $X \not\rightarrow Y$ if $|Y| > 1$. From a practical perspective, cumulating traversals on an FD-tree can easily result in heavy loads of computation because a linkage-based data structure like FD-tree is stored in heap memory. So, in the spirit of eliminating redundant traversals, we introduce synergized induction of FDs. An example shows as follows. Briefly, given a non-FD $X \not\rightarrow Y$, if a FD $X' \rightarrow Y'$ contradicts $X \not\rightarrow Y$ i.e. $X' \subseteq X$ and $Y' \subseteq Y$, one can augment FD $X' \rightarrow Y'$ to generate all non-contradicting and non-trivial FDs by analysing $Y'$.

**Example 4.3 (Synergized Induction of FDs)** Let $AC \rightarrow E$ and $AC \rightarrow BE$ be FDs. If $AC \not\rightarrow BDE$ is a non-FD on a given relation, the two FDs cannot be valid. $ABC \rightarrow E$ and $ACD \rightarrow E$ are all non-trivial candidates of valid FDs that
result from augmenting AC → E. ACD → BE, ABC → E, and ACE → B are all non-trivial candidates of valid FDs that result from augmenting AC → BE.

Algorithm 2 performs a synergized induction to update an extended FD-tree given any invalid FDs. It works as follows. Any FD $X' \rightarrow Y'$ where $X' \subseteq X$ and $Y' \subseteq Y$ cannot be satisfied by $r$ given the non-FD $X \not\rightarrow Y$. In steps 20-24, the algorithm only traverses the paths which are subsets of $X$. During a traversal, a node’s invalid RHS $Y'$ is removed if the node is an FD-node and $Y'$ intersects with $Y$ (step 5-10). Although $X' \rightarrow Y'$ cannot form a valid FD in $r$, there are two ways to add another attribute $A \in R$ to $X'$ such that $X'A$ is the LHS of some candidate FD. (1) We take $A$ outside the union $XY'$ (step 12). Then $X'A \rightarrow Y'$ is non-trivial and has a LHS that is not a subset of $X$. (2) We take $A$ from $Y'$ (step 16) and then $X'A \rightarrow Y' - \{A\}$ is also non-trivial and has a LHS that is not a subset of $X$.

**Algorithm 2** Synergized Induction

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><strong>Input:</strong> A relation schema $R$, an invalid FD $X \not\rightarrow Y$, the root node $root$ of an FD-tree</td>
</tr>
<tr>
<td>2</td>
<td>function $induct(X,Y)$</td>
</tr>
<tr>
<td>3</td>
<td>$induct_{\text{recursive}}(X = {A_1, \ldots, A_n}, Y, root)$</td>
</tr>
<tr>
<td>4</td>
<td>function $induct_{\text{recursive}}(X = {A_i, \ldots, A_n}, Y, current)$</td>
</tr>
<tr>
<td>5</td>
<td>if $current$ is an FD-node then</td>
</tr>
<tr>
<td>6</td>
<td>Let $X'$ be the path leading to $current$</td>
</tr>
<tr>
<td>7</td>
<td>Let $rhs(current)$ be the RHS of $current$</td>
</tr>
<tr>
<td>8</td>
<td>$removed \leftarrow rhs(current) \cap Y$</td>
</tr>
<tr>
<td>9</td>
<td>Remove FD $X' \rightarrow removed$</td>
</tr>
<tr>
<td>10</td>
<td>$rhs(current) \leftarrow rhs(current) - Y$</td>
</tr>
<tr>
<td>11</td>
<td>if $removed \neq \emptyset$ then</td>
</tr>
<tr>
<td>12</td>
<td>for each $A' \in R - (X \cup removed)$ do</td>
</tr>
<tr>
<td>13</td>
<td>$Y' \subseteq removed$ is the minimal RHS of $X'A'$</td>
</tr>
<tr>
<td>14</td>
<td>Add FD $X'A' \rightarrow Y'$ if $</td>
</tr>
<tr>
<td>15</td>
<td>if $</td>
</tr>
<tr>
<td>16</td>
<td>for each $A' \in removed$ do</td>
</tr>
<tr>
<td>17</td>
<td>$removed' \leftarrow removed - {A'}$</td>
</tr>
<tr>
<td>18</td>
<td>$Y' \subseteq removed'$ is the minimal RHS of $X'A'$</td>
</tr>
<tr>
<td>19</td>
<td>Add FD $X'A' \rightarrow Y'$ if $</td>
</tr>
<tr>
<td>20</td>
<td>for each $j \in [i,n]$ do</td>
</tr>
<tr>
<td>21</td>
<td>if $A_j &gt; \max{A' \in R \mid current$ has a child of $A'}$ then</td>
</tr>
<tr>
<td>22</td>
<td>Return</td>
</tr>
<tr>
<td>23</td>
<td>if there is a child $c$ of $current$ with $A_j$ then</td>
</tr>
<tr>
<td>24</td>
<td>$induct_{\text{recursive}}({A_j, \ldots, A_n}, Y, c)$</td>
</tr>
</tbody>
</table>
4.3.3 Dynamic Data Manager

We introduce dynamic data manager that works with extended FD-trees to compute stripped partitions dynamically and efficiently.

DDM pre-computes stripped partitions for all single attributes of a given relation schema. Moreover, DDM maintains an array of dynamic stripped partitions. Dynamic stripped partitions are based on the paths ending at the current controlled level. So, these stripped partitions are refined frequently as the controlled level increases. As we discussed before, ids of nodes in an extended FD-tree index stripped partitions, which are either pre-computed or dynamic (see Example 4.4).

Example 4.4 (Dynamic Data Manager)  Figure 4.4 shows an extended FD-tree over $R = \{A, B, C, D, E, F\}$, with validation and controlled level 3. The DDM contains $\pi_{ABD}$, $\pi_{ACD}$, and $\pi_{ACE}$, indexed by 1, 2, 3, respectively. Since node $F$ resides at level 4, its id is that of its parent node $E$ at the validation level. If the value of an id exceeds $|R|$, the id indexes a stripped partition in the DDM. Here, node $E$’s id (9) corresponds to $\pi_{ACE}$ as $9 - |R| = 3$. The id (7) of node $B$ is inconsistent because it corresponds to $\pi_{ABD}$ as $7 - |R| = 1$. So, if FD $ABC \rightarrow E$ is added to the tree now, node $C$ becomes an FD-node with id 3 (default) instead of 7.
The main task of a DDM is to update dynamic stripped partitions and assign nodes with consistent ids in an FD-tree. Given the nodes at the controlled level, DDM uses the underlying paths to compute new stripped partitions. Algorithm 3 updates a DDM from controlled level $i$ to $j$. For each node at level $j$, the algorithm finds the path to the node, and refines the node’s stripped partition in $A$ using the new attributes in the node’s path. The refined partition is appended to the new array $A'$ (step 10). The new id of the node is the node’s position in $A'$ plus $|R|$ (step 13). Then, the new id is copied to the node’s descendants, ensuring consistency in the extended FD-tree. Any new node that is introduced at the controlled level is not processed by Algorithm 3. Hence, no corresponding stripped partition exists for such a node. Here, the order of its attribute is used as the id of this node (default id).

Algorithm 3 Update Dynamic Data Manager

1: **Input:** A relation $r$ over relation schema $R$, an array $A$ of stripped partitions from level $i$, the set $L$ of all the nodes from level $j$ where $j > i$
2: **Output:** A new array of stripped partitions at level $j$
3: Let $A'$ be an array of size $|L|
4: i ← 1
5: **for each node** $n \in L$ **do**
6: Let $X$ be the path leading to $n$
7: Let $\pi_X = \pi_A$ where $A$ is the attribute of $n$
8: if $n.id > |R|$ then
9: $\pi_X ← A[n.id - |R|]$
10: Let $A'[i] = \pi_X$
11: **for each** $B \in X - X'$ **do**
12: $A'[i] ← \text{refine}(r, A'[i], B)$ \textbf{▷ Algorithm 5}
13: $n.id ← i + |R|$
14: $i ← i + 1$
15: Copy id of node $n$ to its descendants
16: **Return** $A'$

4.3.4 Validation and Refinement

To compute stripped partitions efficiently, we use a domain independent indexing scheme (DIIS). This compresses an input relation into a two-dimensional array. Given relation $r$ over $R$, a DIIS for $A \in R$ is a bijective mapping of the active domain $\text{adom}_r(A)$ to $\{1, \ldots, |\text{adom}_r(A)|\}$. This is easy to compute,
convenient for generating stripped partitions, and validating FDs. Note that
the active domain of \( A \) with respect to \( r \) is just the finite set \( \{ t(A) \mid t \in r \} \).

Validation of an FD \( X \rightarrow Y \) returns a set of non-FDs that cover all RHS
attributes in \( Y \) that are not functionally dependent on the given LHS \( X \). Clas-
sical FD validation, see Section 4.3.2, typically requires multiple runs to find
valid mappings between LHS and RHS values [52], [88]. These mappings are
generated incrementally from the static stripped partitions of the singleton
attributes. Consequently, LHS values are typically computed redundantly.
We now discuss how DDMs improve classical FD validation. With the aim
to avoid redundant computation, we propose Algorithm 4 to validate FDs
using dynamic stripped partitions.

Algorithm 4 FD Validation

1: \textbf{Input:} A relation \( r \) over relation schema \( R \), FD \( X \rightarrow Y \), stripped partition \( \pi_{X'} \) where
\( X' \subseteq X \)
2: \textbf{Output:} Non-FDs that invalidate \( X \rightarrow Y' \) where \( Y' \subseteq Y \)
3: \( \text{non_fds} \leftarrow \emptyset \)
4: \( \text{valid_rhs} \leftarrow Y \)
5: \textbf{for each} \( S \in \pi_{X'} \) \textbf{do}
6: \( \pi \leftarrow \{ S \} \)
7: \textbf{for each} \( A \in X - X' \) \textbf{do}
8: \( \pi \leftarrow \text{refine}(r, \pi, A) \)
9: \textbf{for each} \( \{ t_0, \ldots, t_n \} \in \pi \) \textbf{do}
10: \textbf{for each} \( i \in [1, n] \) \textbf{do}
11: \( \text{invalid_rhs} \leftarrow \{ A \in \text{valid_rhs} \mid t_i(A) \neq t_0(A) \} \)
12: \( \text{valid_rhs} \leftarrow \text{valid_rhs} - \text{invalid_rhs} \)
13: \textbf{if} \( \text{invalid_rhs} \neq \emptyset \) \textbf{then}
14: \( Z \leftarrow \{ A \in R \mid t_0(A) = t_i(A) \} \)
15: \( \text{non_fds} \leftarrow \text{non_fds} \cup \{ Z \not\rightarrow R - Z \} \)
16: \textbf{if} \( \text{valid_rhs} = \emptyset \) \textbf{then}
17: \( \text{Return} \ \text{non_fds} \)
18: \( \text{Return} \ \text{non_fds} \)

Algorithm 4 is a general approach to validating FDs with stripped partitions.
Given relation \( r \) over \( R \) and an FD \( X \rightarrow Y \), a DDM may hold the
stripped partition \( \pi_{X'} \) instead of \( \pi_X \) where \( X' \subseteq X \). In general, it is inefficient
to generate \( \pi_X \) from \( \pi_{X'} \) while validating \( X \rightarrow Y \). Indeed, the computation
of \( \pi_X \) must scan every tuple in \( \pi_{X'} \). This wastes resources if \( X \rightarrow Y \) is not
satisfied by \( r \). In fact, the overhead is substantial if there are many tuples in
\( \pi_{X'} \). Hence, Algorithm 4 only \textit{refines} one set in a stripped partition at a time,
4.3. Dynamic Hybrid Algorithm

using Algorithm 5. It can thus terminate quickly if a given FD is not satisfied. Algorithm 4 reduces redundant computations of $X$-values since $\pi'_{X'}$ is known. Only $(X - X')$-values are processed (step 7-8) if there are valid RHSs. That is, in steps 16 and 17 the algorithm returns a set of non-FDs if there is no valid RHS for $X$.

Algorithm 5 refines stripped partitions. It stores the new equivalence classes in an array. Our data compression scheme eases the allocation of tuples to their new classes in the array, since the index of each class corresponds to some domain value. The algorithm refines every class in the input partition one attribute at a time. When a tuple is allocated to an empty set, we store the set’s position by retrieving the tuple’s projected value on the current attribute (step 8). Recording these positions saves the search for non-empty sets.

Algorithm 5 Refine Stripped partition

1: **Input:** A relation $r$ over relation schema $R$, a subset $\pi'$ of the stripped partition $\pi_X$, an attribute $A \in R$
2: **Output:** A subset of $\pi_{XA}$
3: Let $\text{sets}_{\text{array}} = \{\emptyset, \ldots, \emptyset\}$ where $|\text{sets}_{\text{array}}| = |r|
4: $\text{result} \leftarrow \emptyset$, $\text{ids} \leftarrow \emptyset$
5: for each $S \in \pi$ do
6:      for each $i \in S$ do
7:         if $\text{sets}_{\text{array}}[r[i][A]]$ is empty then
8:            $\text{ids} \leftarrow \text{ids} \cup \{r[i][A]\}$
9:            $\text{sets}_{\text{array}}[r[i][A]] \leftarrow \text{sets}_{\text{array}}[r[i][A]] \cup \{i\}$
10:       for each $id \in \text{ids}$ do
11:          if $|\text{sets}_{\text{array}}[id]| \geq 2$ then
12:             $\text{result} \leftarrow \text{result} \cup \text{sets}_{\text{array}}[id]$
13:             $\text{sets}_{\text{array}}[id] \leftarrow \emptyset$
14:      $\text{ids} \leftarrow \emptyset$
15: Return $\text{result}$

4.3.5 When to Update Stripped Partitions

For enabling a DDM to decide if the stripped partitions need updating, we define the efficiency and inefficiency of a validation level. At each level, the total number of FDs is the sum of the RHS sizes over the nodes at the current level (also see step 13 of Algorithm 6), before FD induction takes place (step 20). The number of valid FDs is counted in the same way (step 13) but after
FD induction. The **efficiency** of the validation level is the ratio of valid FDs over all FDs (including invalid FDs) at the given level. Only actually valid FDs require a scan of all the tuples in a stripped partition. If efficiency is low, more nodes in higher levels may represent invalid FDs. Generating stripped partitions for more invalid FDs is inefficient. The **inefficiency** of a validation level is the proportion of *reusable nodes* over all the FDs that reside in higher levels. Here, a node is *reusable* if it is not a leaf. If inefficiency is high, most FDs in higher levels cannot share stripped partitions. Hence, it is more efficient to directly validate these FDs when their FD node is reached. In summary, updates of stripped partitions are more beneficial when the efficiency at the current validation level is high and the inefficiency is low. Hence, we define the efficiency-inefficiency ratio at the current validation level as the ratio of its efficiency over its inefficiency. Experiments that determine the actual ratio used by DHyFD are in Section 4.4.

**Example 4.5 (Efficiency-inefficiency Ratio)**  
Figure 4.5 shows calculations of the efficiency-inefficiency ratio. After processing level 2 (left tree), node B represents the valid FD $B \rightarrow F$. The efficiency of level 2 (left tree) is $1/1$ since there is only one FD-node at level 2. Both nodes B and C are reusable. They lead to 5 FDs: $ABD \rightarrow C$, $ABD \rightarrow E$, $ACD \rightarrow B$, $ACD \rightarrow F$ and $ACEF \rightarrow B$. So, the inefficiency is $2/5$ and the ratio is $2.5$. After processing level 3 (right tree), the FD $ABD \rightarrow CE$ is valid but the node D in path ABD is not reusable. The other FD
ACD → BF at level 3 (right tree) is not valid. Hence, the efficiency is 1/2. By an induction on invalid FD ACD ⊳ BF, a new path ACDE is constructed. The reusable nodes at level 3 (node D and E) lead to 3 FDs: ACDE → B, ACDE → F, and ACEF → B. So, the inefficiency is 2/3 and the ratio is 0.75.

4.3.6 DHyFD Algorithm

Algorithm 6 implements DHyFD. It starts by initializing the DDM and extended FD-tree (steps 3-4). The controlled and validation levels are tracked by the variables cl and vl, respectively. DHyFD performs the sorted neighborhood pair selection sampling only once at the beginning to extract a diverse selection of non-FDs (step 5). Re-sampling would only cause computational overheads. For example, sampling with an input of 1,000 tuples already compares more than 1 million tuple pairs according to Section 4.4. In steps 14-18, the DDM of DHyFD finds a stripped partition (steps 15-16), and validates the corresponding FDs level by level with Algorithm 4 (line 18). Subsequently, any identified violations of FDs are used to update the extended FD-tree with Algorithm 2 (step 20). Lines 21-25 calculate the efficiency-inefficiency ratio to determine if the DDM should update the stripped partitions (line 27). The iterations continue until no candidate FD is left (step 11).

Note that sorting non-FDs (in step 7 and 19) helps eliminate redundant inductions faster than using non-redundant non-FDs, as demonstrated in Section 4.4. This is explained as follows. Let T be an FD-tree. Suppose an update is processed by a non-FD X ⊳ Y. Now consider another update by a non-FD X' ⊳ A where X' ⊂ X and A' ∈ Y. Here, X' ⊳ A is redundant with respect to X ⊳ A. Hence, no new FDs will be induced if X ⊳ A is applied first. In addition, if X' ⊳ A is applied first, then some of the new FDs can still be eliminated by X ⊳ A, which causes redundant inductions.

We illustrate the steps of Algorithm 6 and Figure 4.3.
Algorithm 6 DHyFD

1. **Input:** A relation \( r \) over relation schema \( R \)
2. **Output:** The left-reduced cover of the FDs satisfied by \( r \)
3. Initialize DDM \( M \) with the stripped partitions of all \( A \in R \)
4. Let \( t \) be an extended FD-tree for the single FD \( \emptyset \rightarrow R \)
5. \( \text{violations} \) is the set of non-FDs extracted by sorted neighborhood pair selection sampling
6. \( \text{violations} = \text{violations} \cup \text{validate}(\text{root}, \{ r \}) \)
7. Sort the non-FDs in descending order by the sizes of their LHSs
8. for each \( X \rightarrow R - X \in \text{violations} \) do \( \text{tree}.\text{induct}(X, R - X) \) \( \triangleright \) Algorithm 2
9. Let \( \text{candidates} \) be the set of nodes at level 1 of \( \text{tree} \)
10. Let \( cl = 1, vl = 1, num\_fds = 0 \)
11. while \( \text{candidates} \neq \emptyset \) do
12. \( \text{violations} = \emptyset \)
13. \( \text{total} = \sum_{n \in \text{candidates}} |\text{rhs}(n)| \)
14. for each \( n \in \text{candidates} \) do
15. if \( \text{node.id} \leq |R| \) then
16. \( \text{node.id} = \arg \text{min}_{A \in R} ||\pi_A|| \)
17. Let \( \pi \) be the stripped partition assigned to \( \text{node} \) by \( M \)
18. \( \text{violations} = \text{violations} \cup \text{validate}(\text{node}, \pi) \) \( \triangleright \) Algorithm 4
19. Sort \( \text{violations} \) in descending order
20. for all \( X \rightarrow R - X \in \text{violations} \) do \( \text{tree}.\text{induct}(X, R - X) \) \( \triangleright \) Algorithm 2
21. \( \text{reusables} = \{ n \in \text{candidates} | n \text{ is not a leaf} \} \)
22. \( \text{num\_new\_fds} = \sum_{n \in \text{candidates}} |\text{rhs}(n)| \)
23. \( \text{num\_fds} = \text{num\_fds} + \text{num\_new\_fds} \)
24. \( \text{efficiency} = \frac{\text{num\_new\_fds}}{\text{total}} \)
25. \( \text{inefficiency} = \frac{|\text{reusables}|}{|\text{tree}| - \text{num\_fds}} \)
26. if \( vl > 1 \) and \( \text{efficiency}/\text{inefficiency} > 3.0 \) then
27. \( cl = vl, \text{Update} \ M \text{ with} \ \text{reusables} \) \( \triangleright \) Algorithm 3
28. \( vl = vl + 1 \)
29. Let \( \text{candidates} \) be the set of nodes of \( \text{tree} \) at level \( vl \)
30. Return \{FDs in tree\}

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>t2</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>t3</td>
<td>6</td>
<td>4</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>t4</td>
<td>2</td>
<td>8</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>t5</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>t6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>t7</td>
<td>2</td>
<td>10</td>
<td>11</td>
<td>2</td>
</tr>
<tr>
<td>t8</td>
<td>12</td>
<td>1</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>t9</td>
<td>13</td>
<td>4</td>
<td>4</td>
<td>14</td>
</tr>
<tr>
<td>t10</td>
<td>15</td>
<td>16</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>t11</td>
<td>17</td>
<td>1</td>
<td>18</td>
<td>1</td>
</tr>
</tbody>
</table>

**Table 4.2:** Toy relation for Example 4.6
Example 4.6 (DHyFD) We demonstrate how DHyFD discovers FDs from the relation $r$ in Table 4.2. For the demonstration purpose, we set the efficiency-inefficiency ratio to 0.1.

Initialization: Initially, the controlled level (cl) is set to 1 and validation level (vl) to 0. The FD $\emptyset \rightarrow ABCD$ is inserted in the empty extended FD-tree, and an array of stripped partitions for single attributes will be computed: 0) $(\pi_A)\{(t_1, t_2, t_6), [t_4, t_5, t_7]\}$, 1) $(\pi_B)\{(t_1, t_6), [t_3, t_5, t_9], [t_8, t_11]\}$, 2) $(\pi_C)\{(t_1, t_2, t_6), [t_3, t_4, t_9], [t_5, t_8, t_{10}]\}$, 3) $(\pi_D)\{(t_1, t_2), [t_8, t_{11}], [t_4, t_7], [t_5, t_6, t_{10}]\}$. In this example, we only let DHyFD run the sorted neighborhood pair selection once to sample non-FDs from $\pi_A, \pi_B, \pi_C, \pi_D$ during the initialization. For instance, pairs $(t_1, t_2), (t_2, t_6), (t_4, t_5),$ and, $(t_5, t_7)$ of $\pi_A$ are used to compute non-FDs. The sampled non-FDs are \{ACD \not\rightarrow B, AC \not\rightarrow BD, A \not\rightarrow BCD, ABC \not\rightarrow D, BD \not\rightarrow AC, B \not\rightarrow ACD, C \not\rightarrow ABD, AD \not\rightarrow BC, D \not\rightarrow ABC\}. In HyFD, sorted neighborhood pair selection is not only used in the initialization but also in iterative steps. The toy example already shows that sampling from stripped partitions of single attributes may generate many redundant non-FDs. For example, only non-FDs ACD \not\rightarrow B, ABC \not\rightarrow D, BD \not\rightarrow AC, and AD \not\rightarrow BC affect the extended FD-tree. Hence, it is inefficient to keep sampling non-FDs during iterative steps.

Iteration 1: DHyFD starts to validate FDs at level 0 of the extended FD-tree. Algorithm 4 uses stripped partition $\pi_\emptyset$ to validate the only FD $\emptyset \rightarrow ABCD$. The tuple pairs $(t_1, t_2), (t_1, t_3)$ identify non-FDs ACD \not\rightarrow B and $\emptyset \not\rightarrow ABD$ as counterexamples to FD $\emptyset \rightarrow ABCD$. Afterwards, all the non-FDs collected from initialization and validation are used in a synergized induction to update the extended FD-tree. This results in the extended FD-tree with FDs $AB \rightarrow C$ and $C \rightarrow A$. The DDM will not update yet as cl only starts from 1 and DHyFD will continue traversing nodes at level 2 of the extended FD-tree. Note that HyFD traverses FD-tree twice to process updates for non-FDs like AD \not\rightarrow BC. Instead, DHyFD only traverses the extended FD-tree once to update all the violations on the RHS of the non-FD.

Iteration 2: At level 1 of the extended FD-tree, node $A$ and FD-node $C$ were
traversed from the previous iteration. So, only FD \( C \rightarrow A \) needs validation. Using pre-computed stripped partition \( \pi_C \), the pair \((t_3, t_4)\) identifies non-FD \( C \not\rightarrow A \) as a counterexample to FD \( C \rightarrow A \). Then, applying the non-FD to the extended FD-tree results a new set of FDs \{\( AB \rightarrow C, BC \rightarrow A, CD \rightarrow A \}\}. There is no need to update DDM since \( cl = 1 \). At this point, all the nodes are assigned with a default id e.g. if a node represents attribute B, then the node has id 2.

**Iteration 3:** At level 2 of the extended FD-tree, three nodes were traversed from the previous iteration and all of them are FD-nodes. Since all the nodes only have default ids, DHyFD applies the stripped partition with the fewest tuples for their validations. For example, \( \pi_B \) is used to validate FD \( AB \rightarrow C \). As a result, only FD \( AB \rightarrow C \) is valid. During validation, the pairs \((t_3, t_9)\) and \((t_5, t_{10})\) identify non-FDs \( BC \not\rightarrow AD \) and \( CD \not\rightarrow AB \). The efficiency of validations at level 2 is 0.33 (1/3). After synergized induction, the new set of FDs is \{\( AB \rightarrow C, BCD \rightarrow A \}\). The inefficiency is 1 (1/1) because only node C is reusable and there is only 1 FD above level 2. So, the efficiency-inefficiency ratio is 0.33, which is greater than the threshold 0.1. Nodes remaining at level 2 are used to compute an array of dynamic stripped partition: 0) \( \pi_{AB} \) \{\( [t_1, t_6] \}\}, 1) \( \pi_{BC} \) \{\( [t_1, t_6], [t_3, t_9] \}\}. Meanwhile, cl becomes 2. At this point, nodes C and D of FD \( BCD \rightarrow A \) are assigned 5 (i.e. 1 + |R|) as their ids. Using the DDM and dynamic stripped partition has two main advantages. Dynamic stripped partitions help improve validation efficiency by avoiding redundant data value computations. Moreover, dynamic stripped partitions decrease in size when cl becomes high. Indeed, tuple pairs that produced counterexamples will disappear from dynamic stripped partitions but only critical tuple pairs remain if violations still occur e.g. \( \pi_{BC} \) contains much less tuples than \( \pi_B \) or \( \pi_C \). In contrast, HyFD has no aim to improve validation efficiency and its random sampling strategy cannot guarantee any improvements on the discovery.

**Iteration 4:** At level 3 of the extended FD-tree, only FD \( BCD \rightarrow A \) requires validation. Since FD-node D has id 5, the dynamic stripped partition \( \pi_{BD} \) (at index 1) can be used for its validation. At the end of this iteration, all FDs \{\( AB \rightarrow C, BCD \rightarrow A \)\} are found to be valid, and there are no more FDs on the next level of
4.4 Experiments

We present experiments and analysis of our new FD discovery algorithm over real-world benchmarks. Our analysis includes measures such as runtime, memory use, row scalability, and column scalability for different interpretations of missing values.

We implemented DHyFD in Visual C++. For comparison we also implemented state-of-the-art algorithms (TANE [52], FDEP [41], HyFD [88]). These algorithms present benchmark performances on datasets with large numbers of rows, or columns, or both [87], [88]. We ran our experiments on an Intel Xeon 3.6 GHz, 256GB RAM, Windows 10 Dell workstation. We used real-world datasets listed in Section 2.6 of Chapter 2.

4.4.1 Parameter Tuning

A major strength of DHyFD is to generate stripped partitions dynamically. They are refined at validation levels with high efficiency and low inefficiency. An important question is what actual efficiency-inefficiency ratio minimizes the runtime of DHyFD. The left of Figure 4.6 shows the time used to discover FDs on the weather dataset for different ratios. The dataset contains 18 columns and more than 260,000 rows. DHyFD performed best when the ratio was around 3.0. For that ratio it discovered the LHS-reduced cover of 68 FDs within 50 seconds. Although the best ratio depends on the dataset, DHyFD performed well on all benchmark data when the ratio was 3.0. As a representative example, the right of Figure 4.6 shows a similar experiment on uniprot512k with 512000 rows and 30 columns. Although the ratio 3.0 is not optimal here, it is very close to the optimal ratio of 2.5 and the performance at ratio 3.0 is satisfying.
4.4.2 Performance on Real-world Data

We conduct experiments that discover FDs for a range of real-world data using DHyFD and other algorithms. In Section 4.3.2 we described how FDEP can be improved over its original proposal with classical FD induction over classical FD-trees [41]. Accordingly, we have two implementations FDEP1 and FDEP2. Both implement the new synergized induction on extended FD-trees, but FDEP2 sorts all of the non-FDs of a relation the same way FDEP does, while FDEP1 computes a non-redundant cover of non-FDs before induction. We can thus provide fair and comprehensive performance reports on these algorithms. The row- and column-based algorithms cannot terminate on all of the datasets within competitive time. Hence, we set the time limit (TL) of the experiments to 1 hour. Note that HyFD also implements our synergized FD induction. For our experiments, we show the number of rows (#R), columns (#C), FDs (#FD) in a left-reduced cover, incomplete rows (#IR), incomplete columns (#IC), missing values (#⊥), and the running time in seconds.

Table 4.4 shows the runtime (in seconds) of all FD discovery algorithms that we have considered. A noteworthy difference is that FDEP2 performs better than the hybrid algorithms on most of the smaller datasets. TANE only performs well on \(fd\_reduced\) because this data is particularly suitable for TANE. TANE traverses the attribute lattice from bottom to top and all the
LHSs of the FDs discovered in \textit{fd\_reduced} only have 3 attributes. As a result, the FDs with short LHSs will be discovered quickly. As our implementation of HyFD uses synergized induction and performs better than the best known bounds, it is a further evidence for the performance gains that synergized induction facilitates. Once there are enough rows and columns, however, the hybrid algorithms perform better. Whenever DHyFD is better than HyFD, then it is significantly better, for example three times as fast on \textit{weather} and \textit{diabetic}. For the cases where HyFD performs better, the difference is not that big.

DHyFD gains performance over HyFD by leveraging more memory whenever new FD discoveries are likely. We report the use of memory by the various algorithms in Table 4.6 and 4.7. In brief, TANE uses huge memory even on small data like \textit{horse} but does not gain performance. Meanwhile, DHyFD only uses more memory when it is rational, as measured by our efficiency-inefficiency ratio. On \textit{PDBX}, only an extremely small number of FDs is satisfied by a large number of tuples, so non-FDs that derive true FDs can be sampled easily. DHyFD shows similar efficient time and memory use as HyFD, which proves that the efficiency-inefficiency ratio also suggests correctly to DHyFD that more memory usage will not improve running time further. On other data, DHyFD outperforms HyFD by better use of memory for FD validation and non-FD extraction. In fact, DHyFD beats HyFD in many cases, such as datasets like \textit{weather}, \textit{lineitem} and \textit{uniprot512k} that contain only a small number of FDs that are randomly spread over the entire FD-tree, and datasets like \textit{diabetic} that are highly dimensional and contain a large number of FDs. Lastly, compared to TANE, DHyFD uses much less memory.

\textbf{FD induction.} We described how FDEP can be improved over its original proposal [41] in Section 4.3.2. To stress our comments, we provide three different implementations FDEP, FDEP1 and FDEP2 to demonstrate the improvements empirically. FDEP2 is the fully optimized implementation which
performs synergized inductions over an extended FD-tree with all the non-FDs being ordered. FDEP2 also implements synergized induction over an extended FD-tree but it additionally eliminates redundant non-FDs before inductions. FDEP implements the FD induction with a classical FD-tree introduced in [41]. To properly compare FDEP with FDEP2, FDEP also orders non-FDs the same way as FDEP2 does. Table 4.5 reports the runtime of our three implementations on the benchmark datasets. Note that it is sufficient to only consider null = null semantics. The results show strong evidences that our proposed techniques lead to major improvements on runtime. FDEP2 outperforms other implementations as well as the best known implementation. Comparing FDEP2 with FDEP demonstrates advantages of synergized induction and extended FD-tree over classical induction and FD-tree. Comparing FDEP2 with FDEP1 demonstrates the efficiency of ordering non-FDs over eliminating redundancies in non-FDs. Note that one rare case shows

<table>
<thead>
<tr>
<th>Dataset</th>
<th>#R</th>
<th>#C</th>
<th>#FD</th>
<th>TANE T</th>
<th>FDEP2 T</th>
<th>HyFD T</th>
<th>DHyFD T</th>
<th>Best Known [88]</th>
</tr>
</thead>
<tbody>
<tr>
<td>iris</td>
<td>150</td>
<td>5</td>
<td>4</td>
<td>0.001</td>
<td>0.002</td>
<td>0.0001</td>
<td>0.0001</td>
<td>0.1</td>
</tr>
<tr>
<td>balance</td>
<td>625</td>
<td>5</td>
<td>1</td>
<td>0.002</td>
<td>0.024</td>
<td>0.001</td>
<td>0.0001</td>
<td>0.1</td>
</tr>
<tr>
<td>chess</td>
<td>28056</td>
<td>7</td>
<td>1</td>
<td>0.154</td>
<td>47.942</td>
<td>0.017</td>
<td>0.017</td>
<td>0.2</td>
</tr>
<tr>
<td>abalone</td>
<td>4177</td>
<td>9</td>
<td>137</td>
<td>0.029</td>
<td>1.191</td>
<td>0.03</td>
<td>0.017</td>
<td>0.2</td>
</tr>
<tr>
<td>nursery</td>
<td>12960</td>
<td>9</td>
<td>1</td>
<td>0.241</td>
<td>13.684</td>
<td>0.011</td>
<td>0.01</td>
<td>0.5</td>
</tr>
<tr>
<td>cancer</td>
<td>699</td>
<td>11</td>
<td>46</td>
<td>0.044</td>
<td>0.048</td>
<td>0.02</td>
<td>0.009</td>
<td>0.2</td>
</tr>
<tr>
<td>bridges</td>
<td>108</td>
<td>13</td>
<td>142</td>
<td>0.03</td>
<td>0.005</td>
<td>0.004</td>
<td>0.003</td>
<td>0.1</td>
</tr>
<tr>
<td>echo</td>
<td>132</td>
<td>13</td>
<td>527</td>
<td>0.01</td>
<td>0.006</td>
<td>0.003</td>
<td>0.002</td>
<td>0.1</td>
</tr>
<tr>
<td>adult</td>
<td>48842</td>
<td>14</td>
<td>78</td>
<td>22.491</td>
<td>129.174</td>
<td>0.279</td>
<td>0.215</td>
<td>1.1</td>
</tr>
<tr>
<td>letter</td>
<td>20000</td>
<td>17</td>
<td>61</td>
<td>208.67</td>
<td>47.4</td>
<td>6.96</td>
<td>2.035</td>
<td>3.4</td>
</tr>
<tr>
<td>ncvoter1k</td>
<td>1000</td>
<td>19</td>
<td>758</td>
<td>0.444</td>
<td>0.216</td>
<td>0.046</td>
<td>0.029</td>
<td>0.4</td>
</tr>
<tr>
<td>hepatitis</td>
<td>155</td>
<td>20</td>
<td>8250</td>
<td>9.851</td>
<td>0.153</td>
<td>0.174</td>
<td>0.189</td>
<td>0.6</td>
</tr>
<tr>
<td>horse</td>
<td>368</td>
<td>29</td>
<td>128727</td>
<td>130.527</td>
<td>3.334</td>
<td>4.728</td>
<td>2.595</td>
<td>7.1</td>
</tr>
<tr>
<td>plista</td>
<td>1000</td>
<td>63</td>
<td>178152</td>
<td>TL</td>
<td>13.894</td>
<td>19.203</td>
<td>15.403</td>
<td>21.7</td>
</tr>
<tr>
<td>flight</td>
<td>1000</td>
<td>109</td>
<td>982631</td>
<td>TL</td>
<td>9.04</td>
<td>37.064</td>
<td>9.934</td>
<td>53.4</td>
</tr>
<tr>
<td>fd</td>
<td>250000</td>
<td>30</td>
<td>89571</td>
<td>8.084</td>
<td>TL</td>
<td>201.005</td>
<td>158.94</td>
<td>41.1</td>
</tr>
<tr>
<td>weather</td>
<td>262920</td>
<td>18</td>
<td>918</td>
<td>TL</td>
<td>TL</td>
<td>332.734</td>
<td>49.839</td>
<td>N/a</td>
</tr>
<tr>
<td>diabetic</td>
<td>101766</td>
<td>30</td>
<td>40195</td>
<td>TL</td>
<td>TL</td>
<td>2864.84</td>
<td>847.582</td>
<td>N/a</td>
</tr>
<tr>
<td>pdb</td>
<td>17305799</td>
<td>13</td>
<td>68</td>
<td>TL</td>
<td>TL</td>
<td>95.893</td>
<td>100.906</td>
<td>240</td>
</tr>
<tr>
<td>lineitem</td>
<td>6001215</td>
<td>16</td>
<td>3984</td>
<td>TL</td>
<td>TL</td>
<td>1352.87</td>
<td>1047.44</td>
<td>2340</td>
</tr>
<tr>
<td>uniprot512k</td>
<td>512000</td>
<td>30</td>
<td>3703</td>
<td>TL</td>
<td>TL</td>
<td>184.573</td>
<td>75.442</td>
<td>N/a</td>
</tr>
</tbody>
</table>

**Table 4.3:** Runtime (in seconds) of discovering FDs from benchmarks under null = null semantics
4.4. Experiments

<table>
<thead>
<tr>
<th>Dataset</th>
<th>#FDs</th>
<th>#IR</th>
<th>#IC</th>
<th>#⊥</th>
<th>TANE</th>
<th>FDEP1</th>
<th>FDEP2</th>
<th>HyFD</th>
<th>DHyFD</th>
</tr>
</thead>
<tbody>
<tr>
<td>cancer</td>
<td>47</td>
<td>16</td>
<td>1</td>
<td>16</td>
<td>0.042</td>
<td>0.094</td>
<td>0.028</td>
<td>0.017</td>
<td>0.008</td>
</tr>
<tr>
<td>bridges</td>
<td>128</td>
<td>38</td>
<td>9</td>
<td>77</td>
<td>0.021</td>
<td>0.006</td>
<td>0.001</td>
<td>0.003</td>
<td>0.003</td>
</tr>
<tr>
<td>echo</td>
<td>355</td>
<td>71</td>
<td>12</td>
<td>132</td>
<td>0.007</td>
<td>0.007</td>
<td>0.001</td>
<td>0.002</td>
<td>0.001</td>
</tr>
<tr>
<td>ncvoter</td>
<td>1249</td>
<td>1000</td>
<td>5</td>
<td>2863</td>
<td>0.06</td>
<td>0.478</td>
<td>0.052</td>
<td>0.017</td>
<td>0.008</td>
</tr>
<tr>
<td>hepatitis</td>
<td>4513</td>
<td>75</td>
<td>15</td>
<td>167</td>
<td>2.136</td>
<td>0.095</td>
<td>0.037</td>
<td>0.085</td>
<td>0.085</td>
</tr>
<tr>
<td>horse</td>
<td>69267</td>
<td>294</td>
<td>21</td>
<td>1605</td>
<td>10.377</td>
<td>1.06</td>
<td>0.445</td>
<td>1.227</td>
<td>0.791</td>
</tr>
<tr>
<td>plista</td>
<td>132554</td>
<td>996</td>
<td>34</td>
<td>23317</td>
<td>3082.23</td>
<td>27.863</td>
<td>2.967</td>
<td>8.801</td>
<td>7.44</td>
</tr>
<tr>
<td>flight</td>
<td>4749681</td>
<td>1000</td>
<td>69</td>
<td>51938</td>
<td>79.568</td>
<td>80.786</td>
<td>4.46</td>
<td>22.618</td>
<td>9.074</td>
</tr>
<tr>
<td>uniprot1k</td>
<td>1874233</td>
<td>1000</td>
<td>212</td>
<td>179129</td>
<td>110.051</td>
<td>32.821</td>
<td>2.721</td>
<td>3.419</td>
<td>2.998</td>
</tr>
<tr>
<td>weather</td>
<td>6582</td>
<td>157895</td>
<td>12</td>
<td>418580</td>
<td>TL</td>
<td>TL</td>
<td>TL</td>
<td>355.11</td>
<td>133.881</td>
</tr>
<tr>
<td>diabetic</td>
<td>144023</td>
<td>100723</td>
<td>7</td>
<td>192849</td>
<td>TL</td>
<td>TL</td>
<td>TL</td>
<td>2444.52</td>
<td>721.609</td>
</tr>
<tr>
<td>uniprot512k</td>
<td>21176</td>
<td>512000</td>
<td>19</td>
<td>3759296</td>
<td>TL</td>
<td>TL</td>
<td>TL</td>
<td>30.149</td>
<td>41.332</td>
</tr>
<tr>
<td>pdb</td>
<td>101</td>
<td>683410</td>
<td>6</td>
<td>2035242</td>
<td>TL</td>
<td>TL</td>
<td>TL</td>
<td>127.089</td>
<td>166.445</td>
</tr>
</tbody>
</table>

Table 4.4: Runtime (in seconds) of discovering FDs from benchmarks under \(null \neq null\) semantics

that the classical method is better i.e. abalone. This is mainly because excessive labelings in an FD-tree sometimes do help prune a search space. However, such situation is rare and improvements are less significant. In general, excessive labelings just create huge computation overhead e.g. FDEP takes almost 200 more seconds than FDEP to perform the same FD induction over adult dataset.

Null semantics. As in practice, many benchmarks contain missing values. Different interpretations of missing values cause differences in discovery algorithms and performance. We report results on the most common semantics that treats missing values just like any other value (\(null = null\)). Results on the semantics where each missing value is treated as a unique value (\(null \neq null\)) are reported in the Table 4.4. In brief, \(null \neq null\) tends to exhibit more FDs and, hence, longer runtime, especially on larger datasets.

The performance of the algorithms is similar to \(null = null\). However, FDEP is fastest on some smaller datasets for \(null \neq null\): bridges, hepatitis and horse.

For weather, diabetic and pdb, the ranking is the same: DHyFD is again the fastest by far on the former two, while HyFD is marginally faster on pdb. On uniprot512k, HyFD is marginally faster than DHyFD under \(null \neq null\).

In summary, DHyFD improves state-of-the-art. It performs well on data
Chapter 4. Ranked Syntactic and Semantic Data Profiles for Functional Dependencies

<table>
<thead>
<tr>
<th>Dataset</th>
<th>#R</th>
<th>#C</th>
<th>#FD</th>
<th>FDEP</th>
<th>FDEP1</th>
<th>FDEP2</th>
<th>Best Known [88]</th>
</tr>
</thead>
<tbody>
<tr>
<td>iris</td>
<td>150</td>
<td>5</td>
<td>4</td>
<td>0.002</td>
<td>0.002</td>
<td>0.002</td>
<td>0.1</td>
</tr>
<tr>
<td>balance</td>
<td>625</td>
<td>5</td>
<td>1</td>
<td>0.031</td>
<td>0.04</td>
<td>0.024</td>
<td>0.2</td>
</tr>
<tr>
<td>chess</td>
<td>28056</td>
<td>7</td>
<td>1</td>
<td>50.192</td>
<td>94.13</td>
<td>47.942</td>
<td>125.5</td>
</tr>
<tr>
<td>abalone</td>
<td>4177</td>
<td>9</td>
<td>137</td>
<td>0.785</td>
<td>2.794</td>
<td>1.191</td>
<td>3.8</td>
</tr>
<tr>
<td>nursery</td>
<td>12960</td>
<td>9</td>
<td>1</td>
<td>23.415</td>
<td>26.205</td>
<td>13.684</td>
<td>46.8</td>
</tr>
<tr>
<td>cancer</td>
<td>699</td>
<td>11</td>
<td>46</td>
<td>0.127</td>
<td>0.09</td>
<td>0.048</td>
<td>0.5</td>
</tr>
<tr>
<td>bridges</td>
<td>108</td>
<td>13</td>
<td>142</td>
<td>0.011</td>
<td>0.007</td>
<td>0.008</td>
<td>0.2</td>
</tr>
<tr>
<td>echo</td>
<td>132</td>
<td>13</td>
<td>527</td>
<td>0.007</td>
<td>0.009</td>
<td>0.006</td>
<td>0.2</td>
</tr>
<tr>
<td>adult</td>
<td>48842</td>
<td>14</td>
<td>78</td>
<td>311.365</td>
<td>278.591</td>
<td>129.174</td>
<td>860.2</td>
</tr>
<tr>
<td>letter</td>
<td>20000</td>
<td>17</td>
<td>61</td>
<td>73.718</td>
<td>130.414</td>
<td>47.4</td>
<td>291.3</td>
</tr>
<tr>
<td>ncvoter1k</td>
<td>1000</td>
<td>19</td>
<td>758</td>
<td>0.384</td>
<td>0.551</td>
<td>0.216</td>
<td>1.1</td>
</tr>
<tr>
<td>hepatitis</td>
<td>155</td>
<td>20</td>
<td>8250</td>
<td>0.532</td>
<td>0.158</td>
<td>0.153</td>
<td>0.8</td>
</tr>
<tr>
<td>horse</td>
<td>368</td>
<td>29</td>
<td>128727</td>
<td>4.985</td>
<td>4.607</td>
<td>3.334</td>
<td>7.2</td>
</tr>
<tr>
<td>plista</td>
<td>1000</td>
<td>63</td>
<td>178152</td>
<td>35.985</td>
<td>17.945</td>
<td>13.894</td>
<td>26.9</td>
</tr>
<tr>
<td>flight</td>
<td>1000</td>
<td>109</td>
<td>982631</td>
<td>16.134</td>
<td>21.28</td>
<td>9.04</td>
<td>53.4</td>
</tr>
</tbody>
</table>

Table 4.5: Runtime (in seconds) of different implementations of FDEP

with more columns and rows, making effective use of conservatively more main memory to discover FDs more quickly. Specialized algorithms outperform hybrids on datasets for which they are designed. Our optimization of FDEP is effective on datasets with few rows and many columns.

4.4.3 Scalability Test

We explore the row- and column-scalability of TANE, FDEP, and both hybrid algorithms. Qualitative and quantitative experiments show how the row and column numbers impact the performance of these algorithms. The qualitative experiments measure the performance of all algorithms on fragments of weather and diabetic with varying numbers of rows and columns. The quantitative experiments show which algorithm performs best on which fragment.

Quantitative Experiments. Each mark in Figure 4.7 and 4.8 represents a data fragment where the numbers of rows and columns are the values at its horizontal and vertical coordinates, respectively. The color of a mark denotes the algorithm that performed the fastest. FDEP wins consistently on the left of both charts. As columns increase, FDEP gains more advantage. For example, in Figure 4.8, FDEP performs worse on data with 10,000 rows
4.4. Experiments

Figure 4.7: Best performers on weather datasets

and 20 columns, but better on data with 10,000 rows and 30 columns. So, FDEP scales well on columns but poorly on rows. DHyFD wins when more rows and columns are present. There are only few fragments where HyFD performs better than DHyFD, and the differences are small in these cases. This is mainly due to the random performance of the sampling method in HyFD. TANE performs poorly because it is targeted at data with FDs that have shorter LHSs, which happens only rarely on real-world data.

Qualitative Experiments. Figure 4.9 and 4.10 show how much one algorithm performs better than the others on changing number of rows or columns in the same dataset. Figure 4.9 displays the results of our row scalability test of the benchmark algorithms and DHyFD. We ran FD discovery on weather by selecting from 1,000 to 260,000 rows, incremented by 1,000 rows at a time. The time of TANE and FDEP dramatically increases whenever many rows exist. When a dataset exceeds approximately 10,000 rows, TANE and FDEP are not feasible. HyFD suffered a significant performance loss when 211,000 rows were present. Instead, DHyFD shows smooth row scalability.

Figure 4.10 displays the results of our column scalability test on diabetic. For comparing the performance of all algorithms we only selected 10,000
Chapter 4. Ranked Syntactic and Semantic Data Profiles for Functional Dependencies

Figure 4.8: Best performers on diabetic datasets

Figure 4.9: Row scalability test on weather datasets
4.4. Experiments

Figure 4.10: Column scalability test on diabetic datasets

rows. TANE performs well when there are less than 15 columns. If there are more than 41 columns, the time of HyFD increases significantly. As we can tell from the second vertical axis (on the right) in Figure 4.10, this change is caused by the doubling of the valid FDs. That means, HyFD requires much more time to validate FDs. In contrast, DHyFD deals with the same situation more smoothly. This demonstrates a huge improvement achieved by the new validation method and the DDM. Lastly, FDEP and DHyFD perform similarly on datasets with few rows. With more columns the performance of FDEP improves over that of DHyFD. This is because FDEP saves substantial FD validation time when more valid FDs are exhibited.

Figure 4.12 shows the results of a row scalability test on ncvoter dataset with a fixed number of 20 columns, and Figure 4.11 shows the results of a column scalability test on ncvoter with a fixed number of 10,000 rows. Similar to the results reported in Figure 4.9 and 4.10, DHyFD scales better than HyFD in terms of rows and columns. In terms of row scalability, TANE is not competitive for even smaller numbers of rows. In terms of column scalability, FDEP performs very similar to DHyFD.
FIGURE 4.11: Column scalability test on \textit{ncvoter} datasets

FIGURE 4.12: Row scalability test on \textit{ncvoter} datasets
### Memory Consumption

Table 4.6 and 4.7 show how much memory (in MB) each of the algorithms consumed on each of the datasets under different null semantics.

**Memory-time Tradeoff.** We conducted additional experiments to show how DHyFD leverages the tradeoff between performance and memory. We quantify on the performance gain and additional memory use by DHyFD over HyFD. The *performance increase rate* (PIR) is the difference in run time of HyFD and DHyFD over that of HyFD, and the *memory increase rate* (MIR) is the difference in memory use of DHyFD and HyFD over that of DHyFD on a dataset. Figure 4.13 and Figure 4.14 illustrate how the numbers of rows and columns in a dataset affect the memory use results in solid performance gains. With more rows or columns, the PIRs and MIRs are typically getting closer.
<table>
<thead>
<tr>
<th>Dataset</th>
<th>File Size</th>
<th>Memory Consumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>cancer</td>
<td>0.02</td>
<td>4 1 1 0.79 0.88</td>
</tr>
<tr>
<td>bridges</td>
<td>0.006</td>
<td>2 0.71 0.76 0.73 0.76</td>
</tr>
<tr>
<td>echo</td>
<td>18.27</td>
<td>1 0.93 0.73 0.73 0.73</td>
</tr>
<tr>
<td>ncvoter1k</td>
<td>0.15</td>
<td>3 2 2 2 2</td>
</tr>
<tr>
<td>hepatitis</td>
<td>0.008</td>
<td>62 2 3 6 8</td>
</tr>
<tr>
<td>horse</td>
<td>0.02</td>
<td>293 60 34 37 43</td>
</tr>
<tr>
<td>fd</td>
<td>67.95</td>
<td>328 N/a 170 181</td>
</tr>
<tr>
<td>plista</td>
<td>0.56</td>
<td>N/a 280 207 193 239</td>
</tr>
<tr>
<td>flight</td>
<td>0.55</td>
<td>N/a 1006 678 693</td>
</tr>
<tr>
<td>weather</td>
<td>16.94</td>
<td>N/a N/a 678 1229</td>
</tr>
<tr>
<td>diabetic</td>
<td>12.05</td>
<td>N/a N/a N/a 1331 2765</td>
</tr>
<tr>
<td>pdb</td>
<td>1218.56</td>
<td>N/a N/a N/a 13414.4 15872</td>
</tr>
<tr>
<td>lineitem</td>
<td>1024</td>
<td>N/a N/a N/a 2662.4 27648</td>
</tr>
<tr>
<td>uniprot512k</td>
<td>631</td>
<td>N/a N/a N/a 4198.4 5222.4</td>
</tr>
</tbody>
</table>

Table 4.7: Memory consumption (MB) of FD discovery on benchmarks under null ≠ null semantics

Figure 4.13: Memory consumption (MB) of FD discoveries on ncvoter datasets with increasing numbers of rows
Recall that we have used a uniform efficiency-inefficiency ratio (i.e. 3) across all benchmarks. On the dataset `lineitem`, DHyFD consumes much more memory than HyFD for this ratio. The main reason is that FDs of the dataset cause much data redundancy. Hence, even after several iterations, many dynamic stripped partitions are still of similar size as the original dataset. Unsurprisingly, a uniform setting is not best for every situation. However, the efficiency-inefficiency ratio by DHyFD makes it possible to balance runtime efficiency with memory consumption, to suit application needs. Higher ratios consume less memory but slow down runtime, and vice versa. This is illustrated in Figure 4.15 on `lineitem`. With a ratio of 60 we obtain a better runtime performance than HyFD with a very similar consumption of main memory.

### 4.5 Covers of Syntactic Profiles

Previous work has represented the output of FD discovery algorithms as LHS-reduced covers. Table 4.8 contains the results of applying standard algorithms to the benchmark data for the computation of canonical covers from LHS-reduced ones [77]. The table shows the number of FDs in a LHS-reduced
Figure 4.15: Balancing time and memory tradeoff with the efficiency-inefficiency ratio on lineitem

cover (|L-r|), the total number of attributes in a LHS-reduced cover (|L-r|), the number of FDs in a canonical cover (|Can|), the total number of attributes in a canonical cover (|Can|), the percentage of the ratios |Can|/|L-r| (%Size) and |Can|/|L-r| (%Cardinality), and the time in seconds to compute a canonical from the LHS-reduced cover (Time). On average, the canonical covers have about 50% savings in both the numbers of FDs and total numbers of attributes, about 25% savings on smaller datasets (the first ten), and about 70% savings on bigger datasets (the remaining eleven). This makes the outputs of FD discovery algorithms not just clearer, because redundancy is avoided, but also easier to comprehend and process. Our results also demonstrate potential for future improvements of discovery algorithms. The gap between LHS-reduced and canonical covers shows that current algorithms do not prune many redundant FDs. However, efficient pruning based on the transitivity rule of FDs (X → Y and Y → Z imply X → Z) is challenging.
| Dataset      | |L-r| | |L-r|| |Can| | |Can|| %Size | %Card | Time |
|--------------|----------------|------|----------------|----------------|------------|------|----|------|----------------|--------|--------|------|
| iris         | 4              | 16   | 4              | 16             | 100        | 100   | >0 |
| balance      | 1              | 5    | 1              | 5              | 100        | 100   | >0 |
| chess        | 1              | 7    | 1              | 7              | 100        | 100   | >0 |
| abalone      | 137            | 715  | 41             | 217            | 30         | 30    | 0.001 |
| nursery      | 1              | 9    | 1              | 9              | 100        | 100   | >0 |
| cancer       | 46             | 214  | 39             | 184            | 85         | 86    | >0 |
| bridges      | 142            | 669  | 65             | 337            | 46         | 50    | 0.002 |
| echo         | 527            | 2322 | 93             | 392            | 18         | 17    | 0.012 |
| adult        | 78             | 495  | 42             | 267            | 54         | 54    | 0.001 |
| letter       | 61             | 786  | 61             | 786            | 100        | 100   | >0 |
| ncvoter1k    | 758            | 3754 | 185            | 927            | 24         | 25    | 0.023 |
| hepatitis    | 8250           | 54821| 2204           | 14718          | 27         | 27    | 0.927 |
| horse        | 128727         | 1045762| 34053         | 267385         | 26         | 26    | 81.85 |
| fd-reduced   | 89571          | 358238| 1550          | 6203           | 2          | 2     | 79.46 |
| plista       | 178152         | 1397038| 22680         | 166963         | 13         | 12    | 276.35 |
| flight       | 982631         | 6106725| 83496         | 520623         | 8          | 9     | 19996 |
| weather      | 918            | 7219 | 514            | 4061           | 56         | 56    | 0.015 |
| diabetic     | 40195          | 464871| 32689         | 378546         | 81         | 81    | 9.14 |
| pdb          | 68             | 157  | 19             | 58             | 28         | 37    | 0 |
| lineitem     | 3984           | 24927| 679            | 4241           | 17         | 17    | 0.6 |
| uniprot512k  | 3703           | 23530| 1677           | 11179          | 45         | 48    | 0.104 |

Table 4.8: Properties of left-reduced & canonical covers
4.6 Relevance Rank of FD

We provide a quantitative and qualitative analysis of applying our relevance measure of FDs to the canonical covers of our benchmark data. The relevance rank of an FD is defined as follows. Relevance ranks aim to capture how much data redundancies that exist in a given relation.

**Definition 4.2 (Relevance Rank of FD)** Let \( r \) be a relation over relation schema \( R \). An FD \( X \rightarrow A \) over \( R \) is satisfied by \( r \). The relevance rank of the FD with respect to \( r \) is \( \sum_{S \in \pi_{X \rightarrow A}(r)} |S| \).

4.6.1 Quantitative Analysis

Table 4.9 lists how many redundant data value occurrences (see Definition 2.22) in each benchmark. Particularly, for each benchmark, we show the total number of value occurrences (#Value), redundant value occurrences (#Redundancy) and the percentage of redundant value occurrences (%Redundancy). Here, we also handle null makers differently i.e. we can either count or ignore null markers when redundancies are calculated.

Table 4.9 provides the first insight ever on the data redundancy exhibited by the benchmarks, and clearly shows the significance of the measure by sheer volume. Furthermore, the impact of nulls can be large.

Next, for each dataset under different null semantics, we show how FDs of benchmarks distribute over a range of relevance rank percentiles i.e. 0%, 2.5%, 5%, 10%, 15%, 20%, 40%, 60%, 80%, and 100% of the maximum rank of some FD exhibited in the dataset. Namely, in each of the following charts, the x-axis is the ranks calculated according to the percentiles and the y-axis is the number of FDs. For example, in each chart, the first bin indicates the number of FDs that result in 0 redundancy in the dataset; the second bin indicates the number of FDs that result in redundancies which are greater than 0 and not more than 2.5% of the maximum rank. Figure 4.16 is the relevance rank
4.6. Relevance Rank of FD

<table>
<thead>
<tr>
<th>Datasets</th>
<th>#Values</th>
<th>Including nulls</th>
<th>Excluding nulls</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>#Redundancy</td>
<td>%Redundancy</td>
</tr>
<tr>
<td>abalone</td>
<td>37593</td>
<td>67</td>
<td>0.18</td>
</tr>
<tr>
<td>adult</td>
<td>683,788</td>
<td>75718</td>
<td>11.07</td>
</tr>
<tr>
<td>balance</td>
<td>3125</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>chess</td>
<td>196392</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>fd</td>
<td>7500000</td>
<td>2500000</td>
<td>33.33</td>
</tr>
<tr>
<td>iris</td>
<td>750</td>
<td>31</td>
<td>4.13</td>
</tr>
<tr>
<td>letter</td>
<td>340000</td>
<td>6809</td>
<td>2</td>
</tr>
<tr>
<td>lineitem</td>
<td>96019440</td>
<td>11407131</td>
<td>11.88</td>
</tr>
<tr>
<td>nursery</td>
<td>116640</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>breast</td>
<td>7689</td>
<td>706</td>
<td>9.18</td>
</tr>
<tr>
<td>bridges</td>
<td>1404</td>
<td>388</td>
<td>28.13</td>
</tr>
<tr>
<td>weather</td>
<td>4732560</td>
<td>1971104</td>
<td>41.65</td>
</tr>
<tr>
<td>diabetic</td>
<td>3052980</td>
<td>420607</td>
<td>13.78</td>
</tr>
<tr>
<td>echo</td>
<td>1716</td>
<td>375</td>
<td>21.85</td>
</tr>
<tr>
<td>flight</td>
<td>109000</td>
<td>48297</td>
<td>44.31</td>
</tr>
<tr>
<td>hepatitis</td>
<td>3100</td>
<td>1588</td>
<td>51.23</td>
</tr>
<tr>
<td>horse</td>
<td>10304</td>
<td>3703</td>
<td>35.94</td>
</tr>
<tr>
<td>ncvoter1k</td>
<td>19000</td>
<td>2886</td>
<td>15.19</td>
</tr>
<tr>
<td>plista</td>
<td>63000</td>
<td>27024</td>
<td>42.9</td>
</tr>
<tr>
<td>uniprot512k</td>
<td>15360030</td>
<td>1288502</td>
<td>8.39</td>
</tr>
<tr>
<td>pdb</td>
<td>224975387</td>
<td>131743942</td>
<td>58.56</td>
</tr>
</tbody>
</table>

Table 4.9: Data redundancy in numbers and percentages

distributions of FDs in the complete benchmarks. Note that balance, chess, and nursery datasets are not included because they do not satisfy any non-trivial FDs. Figure 4.17 and 4.18 are the relevance rank distributions of FDs in the incomplete benchmarks under null = null and null ≠ null semantics respectively. In addition, each of the charts is labeled with the time of computing the distribution.

As a result, many FDs are ranked within a low percentile of redundant data values (more than 0 and less than 5% of the maximum). Data stewards may first focus on the other FDs, including those that are ranked higher, and those that do not cause any data redundancy because the latter may indicate keys. FDs in the low percentile need to be looked at carefully by domain experts: they could indicate the presence of dirty data or represent FDs that hold only accidentally. For such FDs it is also useful to analyse how many of the redundant data values are null marker occurrences. Although there are differences in the various FD numbers in each of the clusters
4.6.2 Qualitative Analysis

We use ncvoter dataset to perform a qualitative analysis. For illustration, a small snippet of ncvoter is shown in Table 4.10. Among many FDs, the full dataset satisfies:

- $\sigma_1 = \emptyset \rightarrow \text{state}$
- $\sigma_2 = \text{last_name}, \text{zip_code} \rightarrow \text{city}$
- $\sigma_3 = \text{street_address} \rightarrow \text{city}, \text{zip_code}$
- $\sigma_4 = \text{last_name}, \text{gender}, \text{zip_code} \rightarrow \text{name_suffix}$
- $\sigma_5 = \text{voter_id} \rightarrow \text{state}$
- $\sigma_6 = \text{first_name}, \text{street_address} \rightarrow \text{voter_id}$
4.6. Relevance Rank of FD

Figure 4.17: Distributions of canonical FDs’ relevance ranks in incomplete benchmarks (null = null)
Chapter 4. Ranked Syntactic and Semantic Data Profiles for Functional Dependencies

Figure 4.18: Distributions of canonical FDs’ relevance ranks in incomplete benchmarks (null ≠ null)
4.6. Relevance Rank of FD

<table>
<thead>
<tr>
<th>voter_id</th>
<th>first_name</th>
<th>last_name</th>
<th>gender</th>
<th>street_address</th>
<th>city</th>
<th>state</th>
<th>zip_code</th>
</tr>
</thead>
<tbody>
<tr>
<td>131</td>
<td>joseph</td>
<td>cox</td>
<td>m</td>
<td>1108 highland ave</td>
<td>new bern</td>
<td>nc</td>
<td>28562</td>
</tr>
<tr>
<td>131</td>
<td>joseph</td>
<td>cox</td>
<td>m</td>
<td>9 casey rd</td>
<td>new bern</td>
<td>nc</td>
<td>28562</td>
</tr>
<tr>
<td>657</td>
<td>essie</td>
<td>warren</td>
<td>f</td>
<td>105 south st</td>
<td>lasker</td>
<td>nc</td>
<td>27845</td>
</tr>
<tr>
<td>725</td>
<td>lila</td>
<td>morris</td>
<td>f</td>
<td>500 w jefferson st</td>
<td>jackson</td>
<td>nc</td>
<td>27845</td>
</tr>
<tr>
<td>244</td>
<td>sallie</td>
<td>futrell</td>
<td>f</td>
<td>9802 us hwy 258</td>
<td>murfreesboro</td>
<td>nc</td>
<td>27855</td>
</tr>
<tr>
<td>247</td>
<td>herbert</td>
<td>futrell</td>
<td>m</td>
<td>9802 us hwy 258</td>
<td>murfreesboro</td>
<td>nc</td>
<td>27855</td>
</tr>
<tr>
<td>440</td>
<td>barbara</td>
<td>johnson</td>
<td>f</td>
<td>6155 kimesville rd</td>
<td>liberty</td>
<td>nc</td>
<td>27298</td>
</tr>
<tr>
<td>464</td>
<td>albert</td>
<td>johnson</td>
<td>m</td>
<td>6155 kimesville rd</td>
<td>liberty</td>
<td>nc</td>
<td>27298</td>
</tr>
<tr>
<td>265</td>
<td>w</td>
<td>johnson</td>
<td>m</td>
<td>11957 us hwy 158</td>
<td>conway</td>
<td>nc</td>
<td>27820</td>
</tr>
<tr>
<td>272</td>
<td>clyde</td>
<td>johnson</td>
<td>m</td>
<td>8944 us hwy 158</td>
<td>conway</td>
<td>nc</td>
<td>27820</td>
</tr>
<tr>
<td>26</td>
<td>louise</td>
<td>johnson</td>
<td>f</td>
<td>113 gentry st #20</td>
<td>wilkesboro</td>
<td>nc</td>
<td>28659</td>
</tr>
<tr>
<td>42</td>
<td>walter</td>
<td>johnson</td>
<td>m</td>
<td>169 otis brown dr</td>
<td>wilkesboro</td>
<td>nc</td>
<td>28659</td>
</tr>
<tr>
<td>604</td>
<td>christine</td>
<td>davenport</td>
<td>f</td>
<td>1710 matthews rd</td>
<td>robersonville</td>
<td>nc</td>
<td>27871</td>
</tr>
<tr>
<td>751</td>
<td>christine</td>
<td>hurst</td>
<td>f</td>
<td>106 w purvis st</td>
<td>robersonville</td>
<td>nc</td>
<td>27871</td>
</tr>
</tbody>
</table>

Table 4.10: Snippet of ncvoter to illustrate data redundancy

However, no algorithms help data stewards understand which FDs are more relevant than others. Recall that the occurrence of a data value is redundant for a set of constraints whenever every change of this value to a different value at this occurrence incurs a violation of some constraint in $\Sigma$ e.g. Definition 2.22. Hence, the value is fixed for this occurrence given $\Sigma$. For example, changing any occurrence of the state value ‘nc’ will result in the violation of $\sigma_1$. This FD expresses that the state value is constant in the dataset. It is meaningful because the dataset only considers voters from the state ‘nc’. As a consequence, FD $\sigma_1$ causes 1,000 data value occurrences to be redundant. Similarly, FD $\sigma_2$ causes 182, FD $\sigma_3$ causes 162, FD $\sigma_4$ causes 61, FD $\sigma_5$ causes 2, and FD $\sigma_6$ causes no redundant occurrences in ncvoter. For instance, each of the highlighted occurrences in Table 4.10 are redundant due to the FD $\sigma_2$. While highly ranked FDs attract interest from data stewards, low ranked FDs are also interesting. For example, FD $\sigma_5$ has the likely key voter_id on its LHS, and the only violation of the key in the full dataset is illustrated by the first two tuples in Table 4.10. Hence, FDs of low rank can hint at possible occurrences of dirty data. Furthermore, FDs with no redundant occurrences may hint at keys. Indeed, the LHS of $\sigma_6$ constitutes a key on ncvoter. More insight unfolds when we exclude null markers from redundant occurrences. In this case, FD $\sigma_4$ causes only 2 redundant occurrences instead of the 61 when null markers are included. If nearly all redundant data values caused by an FD are null markers, then it is likely that this FD is not relevant.
for the dataset. Our rankings provide data stewards with different ways to analyse the relevance of FDs for applications. One view is to fix a column of interest, and see which minimal LHSs cause how many redundant occurrences in that column. For example, some minimal LHSs that functionally determine city in ncvoter are ranked as follows.

<table>
<thead>
<tr>
<th>minimal LHSs for city</th>
<th>#Redundancy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Including nulls</td>
</tr>
<tr>
<td>last_name, zip_code</td>
<td>158</td>
</tr>
<tr>
<td>middle_name, zip_code</td>
<td>231</td>
</tr>
<tr>
<td>street_address</td>
<td>81</td>
</tr>
<tr>
<td>first_name, zip_code</td>
<td>71</td>
</tr>
<tr>
<td>age, gender, zip_code, full_phone_num</td>
<td>173</td>
</tr>
<tr>
<td>voter_id</td>
<td>2</td>
</tr>
<tr>
<td>last_name, age, full_phone_num, register_date</td>
<td>4</td>
</tr>
<tr>
<td>first_name, last_name, full_phone_num, download_month</td>
<td>2</td>
</tr>
</tbody>
</table>

Here, we list redundancies calculated by counting and ignoring null markers i.e. tuples with null makers on either LHS attributes or city will not be counted for redundancies. Indeed, redundancies caused by FDs that do not involve any nulls on LHS and RHS attributes are strong evidence of the FD pattern, and larger numbers of such redundancies are testimony to the stronger relevance of the FD. Figure 4.19 compares the numbers of FDs that
cause up to a given number of redundancies with (blue) and without (orange) nulls. Over four different fragments of ncvoter with 8k, 16k, 512k, and 1024k tuples, it is interesting to see how these numbers remain stable, and how many FDs with small redundancies are shifted to FDs without redundancies when nulls are excluded from occurrences on LHS and RHS attributes.

As illustrated by our example, data redundancy offers a natural ranking for the outputs of FD discovery algorithms.

4.7 Armstrong Samples

Previous research considered FD sets as the only representation for outputs of FD discovery algorithms. We call these syntactic profiles. Informative Armstrong samples are subsets of the given dataset that satisfy the same FDs as the original dataset does. In data profiling, we call informative Armstrong samples also semantic profiles. We promote the use of semantic profiles in addition to syntactic profiles. In database design, research has shown that semantic profiles are typically small in size [33], [34], and are useful for the acquisition of meaningful FDs [65] and data cleaning [106].

Semantic profiles make the outputs of FD discovery more accessible to a broader audience, since the standard database users cannot be expected to understand what FDs are. In addition, semantic profiles are also helpful to users who do understand what FDs are. In particular, semantic profiles provide real-world sample data. We summarize different ways in which semantic profiles can be computed, and conduct experiments on our benchmark data to provide insight into their computation time and characteristics.

Armstrong relations can be constructed using the non-redundant cover of non-FDs [13], [79] as in Example 4.7.

Example 4.7 (Computing Armstrong Relation)  Let $\Sigma$ be the FD set $\{E \rightarrow D,$
Chapter 4. Ranked Syntactic and Semantic Data Profiles for Functional Dependencies

\( D \rightarrow M \) over \( R = \{ \text{Employee}(E), \text{Department}(D), \text{Manager}(M) \} \). The set of non-redundant non-FDs of \( \Sigma \) is \( \Sigma^{-1} = \{ M \not\rightarrow E, DM \not\rightarrow E, \emptyset \not\rightarrow EDM \} \). So, an Armstrong relation for \( \Sigma \) is given by

<table>
<thead>
<tr>
<th>Employee</th>
<th>Department</th>
<th>Manager</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sheldon</td>
<td>Physics</td>
<td>Siebert</td>
</tr>
<tr>
<td>Howard</td>
<td>Engineering</td>
<td>Siebert</td>
</tr>
<tr>
<td>Raj</td>
<td>Physics</td>
<td>Siebert</td>
</tr>
<tr>
<td>Bernadette</td>
<td>Microbiology</td>
<td>Dan</td>
</tr>
</tbody>
</table>

The agree sets of successive rows are the LHSs in \( \Sigma^{-1} \).

Computing a semantic profile of a given relation only has to extract the rows which agree on the LHSs of the non-FDs. A direct approach to computing non-FDs is to scan all the pairs of rows in an input relation. A semantic profile then consists of the rows that cover the LHSs of all non-FDs by their agree sets. However, the direct approach is limited to relations with sufficiently few rows. For input data with more rows we propose an inductive (indirect) approach. In [13], [79], the authors introduced an inductive method to compute the non-redundant cover of non-FDs, given a set of FDs. So, after discovering the FDs of the input data, we can use the algorithm in [13], [79] to induce non-redundant non-FDs. Finding the rows which agree on the non-FDs is simple.

We have implemented the direct and indirect algorithm for computing semantic samples of the benchmark data. The characteristics of our computations are shown in Table 4.11, in particular the runtime of the direct and indirect algorithm, the number of the non-FDs, and the percentage of rows relative to that of the input data. In the experiments, the time limit was set to 24 hours to show the sizes and computing time for all benchmark data. Some of the experiments do take considerable time (more than 12 hours). This illustrates shortcomings in research on semantic samples for large datasets, and present an avenue for future work. In particular, for input data with more
than 30 columns which exhibit more than 100,000 FDs, the indirect algorithm does not terminate within the time limit, for example on horse, plista, and flight. Importantly, the uniprot dataset has not been processed by any of the algorithms under null = null semantics. However, its Armstrong sample can be computed by the direct approach. This is the first representation for all the FDs that hold on this dataset. No research to date has been able to compute a syntactic profile because there are too many FDs. Such examples are not an exception, as evidenced by hepatitis, plista and flight. They provide real-world evidence that it can be easier to compute semantic samples than syntactic samples. This is not surprising as the number of non-FDs may be significantly smaller than the number of FDs exhibited by the data. Hence, semantic profiles provide a concise representation as they capture all FDs while maintaining a small size.

### Table 4.11: Semantic Profiles of Benchmark Data

<table>
<thead>
<tr>
<th>Dataset</th>
<th>#non-FD</th>
<th>%Sample Size</th>
<th>null = null</th>
<th>null ≠ null</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Direct</td>
<td>Indirect</td>
</tr>
<tr>
<td>abalone</td>
<td>64</td>
<td>2.2</td>
<td>4.921</td>
<td>0.084</td>
</tr>
<tr>
<td>adult</td>
<td>55</td>
<td>0.08</td>
<td>495.984</td>
<td>0.475</td>
</tr>
<tr>
<td>balance</td>
<td>8</td>
<td>1.4</td>
<td>0.059</td>
<td>0.001</td>
</tr>
<tr>
<td>chess</td>
<td>12</td>
<td>0.04</td>
<td>151.565</td>
<td>0.015</td>
</tr>
<tr>
<td>fd</td>
<td>313</td>
<td>0.2</td>
<td>TL</td>
<td>218.97</td>
</tr>
<tr>
<td>iris</td>
<td>10</td>
<td>8</td>
<td>0.003</td>
<td>&gt;0</td>
</tr>
<tr>
<td>letter</td>
<td>81</td>
<td>0.3</td>
<td>190.256</td>
<td>2.067</td>
</tr>
<tr>
<td>lineitem</td>
<td>874</td>
<td>0.02</td>
<td>TL</td>
<td>6646.64</td>
</tr>
<tr>
<td>nursery</td>
<td>16</td>
<td>0.1</td>
<td>41.295</td>
<td>0.015</td>
</tr>
<tr>
<td>cancer</td>
<td>43</td>
<td>3.3</td>
<td>0.131</td>
<td>0.011</td>
</tr>
<tr>
<td>bridges</td>
<td>63</td>
<td>19.4</td>
<td>0.009</td>
<td>0.007</td>
</tr>
<tr>
<td>weather</td>
<td>360</td>
<td>0.1</td>
<td>TL</td>
<td>67.12</td>
</tr>
<tr>
<td>diabetic</td>
<td>9370</td>
<td>1.2</td>
<td>TL</td>
<td>3483.7</td>
</tr>
<tr>
<td>uniprot1k</td>
<td>52558</td>
<td>31.4</td>
<td>15493.2</td>
<td>TL</td>
</tr>
<tr>
<td>echo</td>
<td>84</td>
<td>29.5</td>
<td>0.011</td>
<td>0.021</td>
</tr>
<tr>
<td>flight</td>
<td>2589</td>
<td>48.4</td>
<td>13.184</td>
<td>TL</td>
</tr>
<tr>
<td>hepatitis</td>
<td>608</td>
<td>52.9</td>
<td>0.063</td>
<td>45.716</td>
</tr>
<tr>
<td>horse</td>
<td>3076</td>
<td>47.8</td>
<td>1.783</td>
<td>TL</td>
</tr>
<tr>
<td>ncvoter1k</td>
<td>173</td>
<td>13.9</td>
<td>0.791</td>
<td>0.099</td>
</tr>
<tr>
<td>plista</td>
<td>5038</td>
<td>3.4</td>
<td>11.146</td>
<td>TL</td>
</tr>
<tr>
<td>uniprot512k</td>
<td>586</td>
<td>0.03</td>
<td>TL</td>
<td>102.96</td>
</tr>
<tr>
<td>pdb</td>
<td>33</td>
<td>8.10E-05</td>
<td>TL</td>
<td>220.149</td>
</tr>
</tbody>
</table>
Table 4.11 also shows the size of the semantic profiles. Most of them are under 20% of their input size. Only those datasets which produce more FDs than their number of rows have sizes over 20%. However, these datasets (such as hepatitis, horse and flight) are just small samples of their populations, so they are not representative enough. Instead, the semantic profiles of large datasets are less than 2% e.g. pdb has over 17 million rows in the input but only 14 rows in the semantic profile.

Lastly, we demonstrate the row scalability of semantic profiles. Figure 4.20 shows the sizes of semantic profiles for subsets of ncvoter and weather, respectively. For ncvoter datasets, the input relations range from $10^4$ to $10^6$ rows (with a fixed 20 columns), incremented by $10^4$. For weather, the input relations range from 10,000 to 260,000 rows, incremented by 10,000. The experiments reveal important patterns of semantic samples. By observing the behaviour on varying row numbers, the absolute size of semantic profiles tends to converge but their relative size decreases almost exponentially. By observing the behaviour over the number of FDs and non-FDs exhibited, the sizes of semantic profiles have a strong correlation with the number of FDs in the given relations. Both observations provide further evidence that semantic profiles are useful for data profiling, in particular for larger inputs since their semantic profiles tend to be small and easy to compute.
FIGURE 4.20: Scalability test on sizes of semantic profiles
4.8 Summary

In this chapter, we introduced DHyFD to improve the state-of-the-art [90] on FD discovery. Firstly, we use extended FD-trees and synergized induction to further improve the efficiency of FD inductions. Secondly, we use a validation-based hybrid strategy that can quickly validate an FD with the help of stripped partitions while a general non-FD is extracted whenever the FD is invalidated. Thirdly, a dynamic data manger utilizes our efficiency-inefficiency ratio to control the dynamic computation of stripped partitions. Different choices of thresholds for the ratio make it possible to explore tradeoffs between runtime efficiency and memory consumption.

In our experiments, we compared DHyFD to the state-of-the-art FD discovery algorithms including a row-efficient algorithm (TANE) [52], a column-efficient algorithm (FDEP) [41], and a hybrid algorithm (HyFD) [90]. On average, DHyFD has improved HyFD by over 50% in terms of runtime. Additionally, DHyFD exhibits a much better scalability in terms of the numbers of rows or columns. We have also illustrated how different choices of the efficiency-inefficiency ratio unlock different tradeoffs between runtime efficiency and memory consumption.

Another core contribution to FD discovery but also to data profiling as a whole is that we demonstrate how to use FDs to establish syntactic and semantic profiles of real-world data. We define syntactic profiles to be outputs of discovery algorithms that enumerate the syntactic expressions of the underlying class of dependencies, while semantic profiles are outputs of discovery algorithms in the form of Armstrong samples. For syntactic profiles we propose the first measure for ranking the relevance of FD for the input dataset. Our proposal is to use the arguably most natural measure in terms of the number of redundant data values that an FD causes in the given dataset. Quantitative and qualitative analyses show how the ranks draw attention by
the users to FDs that can be explored effectively for data cleaning. For semantic profiles we computed Armstrong samples for all the benchmark real-world datasets. Armstrong samples provide a common interface for domain experts to comprehend data and data dependencies. In particular, when a dataset is big, an Armstrong sample is typically much smaller, enabling users to either directly inspect or quickly query the sampled data.
Chapter 5

Embedded Unique Constraints

Much work has been done on extending the relational model of data to encompass incomplete information. One focus point has been the clarification of the semantics of integrity constraints within the presence of null markers. We propose a new approach whose semantics relies exclusively on the complete fragments of an incomplete relation. For this purpose, we introduce the class of embedded unique constraints. As the first of two parts for this constraint $E : U$, users specify the completeness requirements of applications in the form of the attribute set $E$, called the embedding of $E : U$. The role of $E$ is to select the sub-relation $r^E$ of tuples from the given relation $r$ which do not exhibit any null marker occurrences on any of the attributes in the embedding $E$. As the second part for the constraint $E : U$, users specify the unique constraint $U \subseteq E$. The role of $U$ is to stipulate a subset of attributes in $E$ which can uniquely identify every tuple in the sub-relation $r^E$. The standard notion of a key over complete relations is a special case of an embedded unique constraint whose embedding $E$ consists of all attributes. SQL’s unique constraints also forms the special case of embedded unique constraints whose embedding coincides with the set of attributes in the unique constraint. In this chapter, we introduce and illustrate embedded unique constraints in Section 5.1, establish axiomatic and algorithmic characterizations of their associated implication problem in Section 5.2, as well as structural and computational characterizations of Armstrong relations in Section 5.3, before analyzing related experiments in Section 5.4. Section 5.5
concludes with a brief summary of this chapter. It is noted that the implementation of embedded unique constraints in standard relational database management systems, their application to the enforcement of entity and referential integrity, as well as their use in query optimization, will be illustrated in Section 6.8.

All the results in this chapter have been reported in an article that has been published in the proceedings of the 36th International Conference on Conceptual Modeling (ER2017) [108]. The article is entitled “Contextual Keys”, and subsequently we found the name embedded unique constraint to be more appropriate.

5.1 Introduction

Keys are core enablers for data management. They are fundamental for understanding the structure and semantics of data. Given a collection of entities, a key is a set of attributes whose values uniquely identify an entity in the collection. For example, a key for a relational table is a set of columns such that no two different rows have matching values in each of the key columns. Keys are essential for many other data models, including semantic models, object models, probabilistic models, XML, RDF, and graphs. They help in many classical areas of data management, including data modeling, database design, indexing, and query optimization. Knowledge about keys enables us to:

- uniquely reference entities across data repositories;
- minimize data redundancy at schema design time to process updates efficiently at run time;
- provide a query optimizer with new access paths that can lead to substantial speedups in query processing;
• allow the database administrator to improve the efficiency of data access via physical design techniques such as data partitioning or creation of indexes and materialized views;

• and provide new insights into application data.

Modern applications raise the importance of keys further. They facilitate the data integration process, help with the detection of duplicates and anomalies, provide guidance in repairing data, and return consistent answers to queries over dirty data. The discovery of keys is one of the core activities in data profiling.

An important and rich area of research is to extend the relational model of data to encompass incomplete information. This is due to the importance of incomplete information for applications. A plethora of different extensions exist, but many are based on the use of a special symbol as placeholder for incomplete information, also known as the null marker. In Chapter 3, we have stated that the semantics of the null marker can vary greatly, for example “unknown at present” [46], “non-existence” [78], “inapplicable” [30], [46], “no information” [112] and “open” [44]. Based on certain interpretation, several extensions of the notion of a key from complete to incomplete relations have been investigated in the research literature. Examples constitute SQL’s primary and candidate keys as well as unique constraints [59], weak and strong keys [66], Codd’s keys [48], possible and certain keys [59], [61], [62], and key sets [68], [101]. Candidate keys are minimal sets of attributes that enable us to uniquely identify tuples in an incomplete relation and where no null markers are permitted to occur in the columns of the key. They are a result of Codd’s principle of entity integrity. The principle has been challenged by several researchers, including Thalheim [101], Levene and Loizou [68], and Köhler et al. [61]. For example, certain keys can uniquely identify rows in a table even though null markers may occur in the key columns. Similarly, for all pairs of distinct tuples there is some key in a key set on which the two tuples have no null marker occurrences and are unique.
Table 5.1 shows an incomplete relation, where \( \perp \) denotes a null marker occurrence. Interestingly, this relation does not satisfy any candidate key, any certain key, nor any key set. It does not satisfy any candidate key as null markers occur in Department and Manager, and there are different tuples with the same value on Employee. The relation violates every certain key since the two null marker occurrences may be replaced by the values Toys and Burns, respectively, resulting in two different tuples that have matching values on all attributes. The relation violates every key set because the first and second tuple are incomplete on Department and on Manager, and have matching values on Employee.

Common to all extended notions of keys is the target of uniquely identifying all tuples in incomplete relations, even tuples with null marker occurrences. The example in Table 5.1 shows that this target cannot always be achieved. In fact, any semantics of a key that depends on the interpretation of null markers can easily become problematic. This holds especially when data is integrated from different sources, which may rely on different interpretations of null markers. Interestingly, SQL’s unique constraint enforces uniqueness only for those tuples of an incomplete relation that are complete on the attributes of the unique constraint. For example, the incomplete relation in Table 5.1 satisfies the unique constraints unique (Employee, Department) and unique (Employee, Manager), but violates the unique constraints unique (Employee) and unique (Department, Manager). This approach sparked our idea of giving up any false hope that tuples can be uniquely identified in the presence of null marker occurrences. In SQL’s unique constraint this given set
of attributes forms the unique constraint itself. That, however, is a requirement that should be relaxed, as the incomplete relation in Table 5.1 illustrates. In fact, unique constraint unique \((\text{Employee, Department})\) can distinguish between the first and third tuple by the values on \text{Employee} already, and does not require values on \text{Department}. Note that uniqueness only holds for the tuples which are complete on \text{Employee} and \text{Department}, and uniqueness does not hold for the tuples that are complete on \text{Employee} only.

Motivated by these examples, we propose the new notion of \textit{embedded unique constraint} (eUC) for incomplete relations. EUCs target the unique identification of those tuples in an incomplete relation that are complete on a user-specified set of attributes. EUCs consist of a pair of attributes in the form \(E : U\) such that \(U \subseteq E\). Note that we sometimes enclose eCUs with parenthesis just for clearance of presentation e.g. eUC \((E : U)\). The user-specified set of attributes \(E\) is called the \textit{embedding} of the eUC, and selects the \textit{scope} of the unique constraint, which is defined as the subset of tuples in a given incomplete relation that are complete on all the attributes of the embedding. The set \(U\) of an eUC \(E : U\) is called the \textit{unique constraint} and uniquely identifies tuples in the scope. For example, \((\{\text{Employee, Department}\}:\{\text{Employee}\})\) and \((\{\text{Employee, Manager}\}: \{\text{Employee}\})\) are both eUCs that are satisfied by the incomplete relation in Table 5.1. Here, both embeddings \{\text{Employee, Department}\} and \{\text{Employee, Manager}\} have the same scope in the incomplete relation, which consists of the first and third tuple, and the unique constraint \{\text{Employee}\} uniquely identifies tuples in this scope. The incomplete relation in Table 5.1 does not satisfy any of the following eUCs: \((\{\text{Employee}\}:\{\text{Employee}\})\), \((\{\text{Employee, Department}\}: \{\text{Department}\})\) and \((\{\text{Employee, Manager}\}: \{\text{Manager}\})\).

In particular, SQL’s unique constraint \(X\) is satisfied by an incomplete relation if and only if the relation satisfies the eUC \(X : X\).

\textbf{Contribution.} Our contributions are summarized as follows.

1. We propose a novel class of integrity constraints for incomplete relational databases, named \textit{embedded unique constraints}. 
2. We characterize the implication problem of eUCs by a finite axiomatization and by a linear-time algorithm. An immediate application of the algorithm is to compute a non-redundant cover of eUCs, thereby minimizing the overhead of enforcing eUCs on relations.

3. We investigate structural and computational properties of Armstrong relations for eUCs, providing a computational tool that aids with the acquisition of eUCs. While the problem of finding an Armstrong relation is precisely exponential, we show that it is linear in the input size when the underlying set of attributes is fixed. Our algorithm is conservative in its use of time and space, as the output Armstrong relation is guaranteed to have a number of tuples that is at most quadratic in the minimum number of tuples required. For transfer into practice, we have implemented our algorithm in a prototype system. Experiments with the prototype system complement our theoretical complexity analysis, and illustrate - on average - how quickly Armstrong relations for eUCs can be computed, how many tuples our output contains, and how many null markers occur in the output. For example, for a fixed schema with 15 attributes, and a set of eUCs with 100 attributes, our algorithm computes an Armstrong relation with 86 tuples and 200 null marker occurrences in about 10 seconds.

5.2 Fundamentals of Embedded Unique Constraints

First of all, we formally introduce a new class of integrity constraints for incomplete relations which we call embedded unique constraint.

Definition 5.1 (Embedded Unique Constraint)  An embedded unique constraint (eUC) over a relation schema $R$ is a statement of the form $(E : U)$ where $U \subseteq E \subseteq R$. The attribute set $E$ is called the embedding of the eUC; $U$ is the unique constraint (UC) of the eUC. A relation $r$ over $R$ satisfies the eUC $(E : U)$,
denoted as \( r \models (E : U) \), if and only if for all \( t, t' \in r^E \), \( t(U) = t'(U) \) implies \( t = t' \).

We call \( r^E \) the scope of \( r \) with respect to the embedding \( E \).

Next we illustrate the notion of eUCs on our running example.

**Example 5.1 (EUC)** The incomplete relation \( r \) in Table 5.1 satisfies the eUC \((ED : E)\) and \((EM : E)\), but it violates the eUC \((ED : D)\) and \((EM : M)\). For example, \( r \) satisfies eUC \((ED : E)\), since the scope of \( r \) with respect to the embedding \( ED \) consists of the first and third tuple of \( r \), and the values of these tuples on \( E \) are different. Similarly, \( r \) does not satisfy \((ED : D)\), since the scope of \( r \) with respect to the embedding \( E \) is the relation \( r \) itself, but the values of the first and second tuples on \( E \) are the same.

For what follows, we require the following concepts. Let \( R \) be a relation schema. We define a partial order \( \sqsubseteq_R \) over \( R \) as \( \{(C_1 : K_1), (C_2 : K_2) \mid C_1 \subseteq C_2 \subseteq R, K_1 \subseteq K_2 \subseteq R\} \). For any \( ((C_1 : K_1), (C_2 : K_2)) \in \sqsubseteq_R \), we write \((C_1 : K_1) \sqsubseteq_R (C_2 : K_2)\), or \((C_1 : K_1) \sqsubseteq_R (C_2 : K_2)\) if \( C_1 \subseteq C_2 \) or \( K_1 \subseteq K_2 \). We may omit the subscript \( R \), if \( R \) is clear from the context. Let \( \Sigma \) be a set of eUCs over \( R \). We define the set \( CL(\Sigma) = \{(E : U) \mid \Sigma \not\models (E : U), U \subseteq E \subseteq R\} \). The set of anti-eUCs of \( \Sigma \) is \( \Sigma^{-1} = \{(E : U) \in CL(\Sigma) \mid \neg\exists(E' : U') \in CL(\Sigma) : (E : U) \sqsubseteq (E' : U')\} \). We also need to re-define *agree sets* in the setting of incomplete information. Let \( r \) be a relation over \( R \). We say \( t_1 \) and \( t_2 \) exactly agree on \((E : U)\) if and only if \( t_1(U) = t_2(U) \) and \( t_1(A) = \bot \lor t_2(A) = \bot \) for all \( A \in R - E \). The *agree set* of \( r \) is \( agr(r) = \{(E : U) \mid t_1, t_2 \text{ exactly agree on } (E : U) \text{ for all distinct } t_1, t_2 \in r\} \).

To reason about eUCs, we introduce the set \( \mathcal{B} \) of inference rules as shown in Table 5.2. Our goal is to show that \( \mathcal{B} \) is a set of sound and complete inference rules for eUCs.

This goal can be realized by using the following syntactic characterization of \( \mathcal{B} \).
Theorem 5.2.1 (Syntactic Characterization of EUC). Let $\Sigma \cup \{(E : U)\}$ be a set of eUCs over a relation schema $R$. $\Sigma \vdash_{\mathfrak{B}} (E : U)$ if only if there is $(E' : U') \sqsubseteq (E : U)$ where $(E' : U') \in \Sigma \cup \{(R : R)\}$.

Proof. Suppose there is $(E' : U') \sqsubseteq (E : U)$ where $(E' : U') \in \Sigma \cup \{(R : R)\}$. For sufficiency, we construct a sequence $\sigma_1 = (E' : U'), \sigma_2 = (E'X : U'Y)$ where $X = E - E'$ and $Y = U - U'$. Therefore, $\Sigma \vdash_{\mathfrak{B}} (E : U)$ because $\sigma_2$ results from $\sigma_1$ by applying the eUC extension axiom. For necessity, suppose $(E' : U') \not\sqsubseteq (E : U)$ for all $(E' : U') \in \Sigma \cup \{(R : R)\}$. Consequently, there is no way to apply any axiom in $\mathfrak{B}$ to $\Sigma \cup \{(R : R)\}$. Therefore, $\Sigma \not\vdash_{\mathfrak{B}} (E : U)$. □

We illustrate the usefulness of Theorem 5.2.1 on our running example.

Example 5.2 (EUC Derivations) Given the set $\Sigma = \{(ED : E),(EM : E)\}$ we can use Theorem 5.2.1 to conclude that there is a derivation of the eUC $(ED : ED)$ and $(EM : EM)$ from $\Sigma$ by $\mathfrak{B}$ since $(ED : E) \sqsubseteq (ED : ED)$ and $(EM : E) \sqsubseteq (EM : EM)$ hold. Similarly, we can use Theorem 5.2.1 to conclude that there is no derivation of eUC $(E : E)$ from $\Sigma$ by $\mathfrak{B}$ since neither $(ED : E) \sqsubseteq (E : E)$ nor $(EM : E) \sqsubseteq (E : E)$ hold.

Lemma 5.2.2 (Total Relation Projection). Let $r$ be a relation over relation schema $R$. If $U \subseteq V \subseteq R$, then $r^V \subseteq r^U$.

Proof. Suppose $U \subseteq V \subseteq R$. Take any $t \in r^V$. For any $A \in V$, $t(A) \neq \bot$. Since $U \subseteq V$, $t(A) \neq \bot$ for any $A \in U$. Therefore, $t \in r^U$ and $r^V \subseteq r^U$. □

Using Theorem 5.2.1, we can establish the following axiomatic characterization.
**Theorem 5.2.3** (Axiomatization of EUCs). \( \mathfrak{B} \) forms a sound and complete axiomatization for the implication of eUCs.

**Proof.** For soundness, suppose \( \Sigma \vdash_{\mathfrak{B}} (E : U) \). By Theorem 5.2.1, there is \( (E' : U') \subseteq (E : U) \) where \( (E' : U') \in \Sigma \cup \{(R : R)\} \). Take any relation \( r \) over \( R \) where \( r \models \Sigma \). We know that any relation over \( R \) satisfies \( (R : R) \). Since \( (E' : U') \in \Sigma \cup \{(R : R)\} \) and \( r \models \Sigma \), thus \( r \models (E' : U') \). Namely, for all \( t_1, t_2 \in r^{E'} \), \( t_1(U') = t_2(U') \) implies \( t_1 = t_2 \). Take any \( t_1, t_2 \in r^E \). On one hand, since \( E' \subseteq E \), by Lemma 5.2.2, \( r^E \subseteq r^{E'} \) and \( t_1, t_2 \in r^E \). On the other hand, since \( U' \subseteq U \), \( t_1(U) = t_2(U) \) implies \( t_1(U') = t_2(U') \). Consequently, since \( t_1(U) = t_2(U) \) implies \( t_1(U') = t_2(U') \) and \( t_1(U') = t_2(U') \) implies \( t_1 = t_2 \), by syllogism, hence \( t_1(U) = t_2(U) \) implies \( t_1 = t_2 \). Therefore, \( \Sigma \models (E : U) \).

For completeness, we prove \( \Sigma \models (E : U) \) implies \( \Sigma \vdash_{\mathfrak{B}} (E : U) \). Assume \( \Sigma \not\models (E : U) \). Next, we show a counterexample as follows.

<table>
<thead>
<tr>
<th></th>
<th>( U )</th>
<th>( E - U )</th>
<th>( R - E )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_1 )</td>
<td>0\ldots 0</td>
<td>0\ldots 0</td>
<td>( \bot \ldots \bot )</td>
</tr>
<tr>
<td>( t_2 )</td>
<td>0\ldots 0</td>
<td>1\ldots 1</td>
<td>( \bot \ldots \bot )</td>
</tr>
</tbody>
</table>

Since \( \Sigma \not\models (E : U) \), by Theorem 5.2.1, there is no \( (E' : U') \subseteq (E : U) \) where \( (E' : U') \in \Sigma \cup \{(R : R)\} \). If \( E' \subseteq E \), then \( U' \not\subseteq U \) and \( t_1(U) \neq t_2(U) \). If \( E' \not\subseteq E \), then \( r^{E'} = \emptyset \). In summary of both cases, \( r \models \Sigma \). However, \( r \not\models (E : U) \) and hence \( \Sigma \not\models (E : U) \), which draws a contradiction to \( \Sigma \models (E : U) \).

Therefore, \( \Sigma \models (E : U) \) implies \( \Sigma \vdash_{\mathfrak{B}} (E : U) \). \( \square \)

We apply Theorem 5.2.3 to our running example.

**Example 5.3 (Implication of EUCs)** Recall that we inferred in Example 5.2 that we can use \( \mathfrak{B} \) to derive eUC \( (ED : ED) \) and \( (EM : EM) \) from \( \Sigma = \{(ED : E), (EM : E)\} \), but we cannot use \( \mathfrak{B} \) to derive the eUC \( (E : E) \) from \( \Sigma \). Based on the soundness of \( \mathfrak{B} \) we can further conclude that eUC \( (ED : ED) \) and \( (EM : EM) \) are implied by \( \Sigma \). Furthermore, based on the completeness of \( \mathfrak{B} \) we can conclude that the eUC \( (E : E) \) is not implied by \( \Sigma \).
The axiomatization can be used to explicitly enumerate all eUCs that are implied by a given set \( \Sigma \). In practice, however, one may not be interested in all eUCs that are implied, but only be interested whether a given eUC \( \varphi \) is implied by \( \Sigma \). In such situation, an explicit enumeration of all implied eUCs is inefficient and does not make good use of the additional input \( \varphi \). For this purpose, we introduce a linear-time algorithmic characterization of the implication problem associated with eUCs. In fact, we use Theorem 5.2.1 to obtain this algorithm.

**Algorithm 7** Implication of EUCs

1. **INPUT:** A set \( \Sigma \cup \{ (E : U) \} \) of eUCs over relation schema \( R \)
2. **OUTPUT:** true, if \( \Sigma \models (E : U) \); false, otherwise
3. for each \( (E' : U') \in \Sigma \cup \{ (R : R) \} \) do
   4. if \( E' \subseteq E \) and \( U' \subseteq U \) then
      5. return true
   6. return false

The correctness of Algorithm 7 follows from Theorem 5.2.1 and Theorem 5.2.3, and the linear-time complexity is easy to observe from the algorithm.

**Theorem 5.2.4** (Complexity of EUC Implication). Algorithm 7 decides the implication problem of eUC \( \Sigma \models (E : U) \) in time \( \mathcal{O}(||\Sigma \cup \{ (R : R) \}||) \).

**Proof.** Assume that checking if an attribute belongs to a set only takes constant time. Time for checking step 4 is in \( \mathcal{O}(|E'| + |U'|) \). Therefore, in total, Algorithm 7 runs in \( \mathcal{O}(||\Sigma \cup \{ (R : R) \}||) \). \(\square\)

We conclude this section by a final example.

**Example 5.4 (Implication Algorithm of EUCs)** Using \( R = EDM \), \( \Sigma = \{ (ED : E), (EM : E) \} \) and \( \varphi = (E : E) \) as input, Algorithm 7 returns false, since there is no \( (C' : K') \in \Sigma \cup \{ (R : R) \} \) such that \( C' \subseteq \{ E \} \) and \( K' \subseteq \{ E \} \).
5.3 Armstrong Relations for Embedded Unique Constraints

EUCs can enforce important application semantics within a database system. However, a fundamental problem is to acquire those eUCs that are meaningful in a given application domain. Database designers usually do not know the domain well and domain experts do not know database constraints. We will now establish computational support for overcoming the communication barrier between designers and experts. As illustrated in Figure 5.1, designers think in terms of an abstract set Σ of eUCs they perceive meaningful. For them to communicate their current perceived understanding to domain experts, we will establish an algorithm that computes from a given set Σ a relation \( r_\Sigma \) that perfectly represents Σ. That is, \( r_\Sigma \) satisfies all eUCs in Σ and violates all eUCs that are not implied by Σ. Relations with this property are known as Armstrong relations [39]. If designers currently perceive an actually meaningful eUC as meaningless, then this eUC will be violated in \( r_\Sigma \). The point is that domain experts will easily notice this violation when some eUC contradicts their domain knowledge. The experts can then alert the designers to this inconsistency with the application semantics, and the designer can include the meaningful eUC in their set Σ. Such process can be repeated until both designers and experts are happy. The other direction, in which one provides computational support for identifying the set Σ of eUCs that hold in a given relation, will be discussed in details in Chapter 6. This direction is
useful as the domain expert may want to change values in an Armstrong relation, or legacy data becomes available to the designers. For the remainder of this section, we will investigate computational and structural properties of Armstrong relations for eUC. We begin with the definition of Armstrong relations for eUCs.

**Definition 5.2 (Armstrong Relations of eUC)** Let $\Sigma$ be a set of eUCs over relation schema $R$. A relation $r$ over $R$ is Armstrong for $\Sigma$ if and only if the following property holds for all eUC $(E, U)$ over $R$: $r \models (E : U)$ if and only if $\Sigma \models (E : U)$. It is worth noticing that one can reduce every instance $\Sigma \models \varphi$ of the implication problem for eUCs to checking whether $\varphi$ holds on $r$ given an Armstrong relation $r$ for $\Sigma$. In fact, if $r$ satisfies $\varphi$, then $\Sigma$ implies $\varphi$; and if $r$ does not satisfy $\varphi$, then $r$ is not implied by $\Sigma$.

**Example 5.5 (Armstrong Relation of eUC)** The relation $r$ in Table 5.1 is Armstrong for the set $\Sigma = \{(ED : E), (EM : E)\}$ over the relation schema $R = EDM$. It is indeed easy to observe that $r$ satisfies $(ED : ED)$ and $(EM : EM)$, which are therefore implied by $\Sigma$. Similarly, $r$ violates $(E : E)$, and $(EDM : DM)$ which are therefore not implied by $\Sigma$.

We say that a class of constraints $\mathcal{C}$ enjoys Armstrong relations if there is an Armstrong relation for every given set of constraints of $\mathcal{C}$.

**Theorem 5.3.1 (Armstrong Relation of EUC).** EUCs enjoy Armstrong relations.

**Proof.** Let $\Sigma$ be a set of eUCs over relation schema $R$. For all $U \subseteq E \subseteq R$, if $\Sigma \not\models (E : U)$, we construct a set $T_{(E:U)} = \{t_1, t_2\}$ where $t_1, t_2 \in r^E$ and $t_1(A_1) = t_2(A_1), t_1(A_2) \neq t_1(A_2), t_1(A_3) = t_2(A_3) = \bot$ for all $A_1 \in U, A_2 \in E - U, A_3 \in R - E$. In addition, we use totally distinct non-null values for the construction $T$ of two distinct eUCs. So, we claim $r = \bigcup_{\Sigma \not\models (E:U)} T_{(E:U)}$ is an Armstrong relation. Take any distinct $t_1, t_2 \in r$. If $t_1, t_2 \in T_{(E:U)}$ where
5.3. Armstrong Relations for Embedded Unique Constraints

Let \( U \subseteq E \subseteq R \), then they form an example for \( \Sigma \not\models (E : U) \); Otherwise, they will not show any violations to any eUC \( (E : U) \) where \( \Sigma \models (E : U) \) because \( t_1, t_2 \) are constructed from totally different non-null values.

Next we would like to characterize the structure of Armstrong relations for eUCs. The following result establishes a necessary and sufficient condition for a given relation to be Armstrong for a given set of eUCs.

**Theorem 5.3.2** (Armstrong Relation of EUC). Let \( \Sigma \) be set of eUCs over relation schema \( R \). A relation \( r \) over \( R \) is Armstrong for \( \Sigma \) if and only if \( \Sigma^{-1} \subseteq \text{agr}(r) \subseteq \text{CL}(\Sigma) \).

**Proof.** \( \Rightarrow \): Suppose \( r \) is an Armstrong relation with respect to \( \Sigma \). 1) Take any \( (E : U) \in \Sigma^{-1} \). Assume \( (E : U) \not\in \text{agr}(r) \). Since \( r \) is an Armstrong relation of \( \Sigma \) and \( (E : U) \in \Sigma^{-1} \), then \( r \not\models (E : U) \). Consequently, by \( (E : U) \in \Sigma^{-1} \) and \( (E : U) \not\in \text{agr}(r) \), there should be at least a pair of distinct \( t_1, t_2 \in r \) such that \( t_1 \) and \( t_2 \) exactly agree on \( E' : U' \) where \( (E : U) \sqsubset (E' : U') \). Namely, \( t_1, t_2 \in r^{E'} , t_1(U') = t_2(U') \), and \( t_1(A) \neq t_2(A) \) for all \( A \in E' - U' \). Since \( (E : U) \) is an anti-eUC, \( (E' : U') \) should not be an anti-eUC because \( (E : U) \sqsubset (E' : U') \). However, \( t_1, t_2 \) exactly agree on \( E' : U' \), which is a contradiction to \( (E : U) \in \Sigma^{-1} \). Therefore, \( (E : U) \in \text{agr}(r) \). 2) Take any \( (E : U) \in \text{agr}(r) \). There are distinct \( t_1, t_2 \in r \) such that \( t_1, t_2 \in r^{E} \), \( t_1(U) = t_2(U) \) and \( t_1(A) \neq t_2(A) \) for all \( A \in E - U \). Since \( t_1 \neq t_2 \), \( \Sigma \not\models (E : U) \). Therefore, \( (E : U) \in \text{CL}(\Sigma) \).

\( \Leftarrow \): Let \( \Sigma^{-1} \subseteq \text{agr}(r) \subseteq \text{CL}(\Sigma) \). We show \( r \models (E : U) \) if and only if \( \Sigma \models (E : U) \) for all \( U \subseteq E \subseteq R \). Suppose \( r \models (E : U) \) where \( U \subseteq E \subseteq R \). 1) For sufficiency, assume \( \Sigma \not\models (E : U) \). Consequently, there exists an anti-eUC \( (E' : U') \in \Sigma^{-1} \) where \( (E : U) \sqsubset (E' : U') \). Since \( \Sigma^{-1} \subseteq \text{agr}(r) \), thus \( (E' : U') \in \text{agr}(r) \). So, there are distinct \( t_1, t_2 \in r \) where \( t_1, t_2 \in r^{E'} \) and \( t_1(U') = t_2(U') \). Moreover, \( t_1, t_2 \) are \( E \)-total and \( t_1(U) = t_2(U) \) because \( (E : U) \sqsubset (E' : U') \). However, since \( t_1 \neq t_2 \), then \( r \not\models (E : U) \), which draws a contradiction to the fact \( r \models (E : U) \). 2) For necessity, suppose \( r \not\models (E : U) \).
So, there exists distinct \( t_1, t_2 \in r \) such that \( t_1, t_2 \) exactly agree on \((E' : U')\) where \((E : U) \subseteq (E' : U')\). So, \((E' : U') \in agr(r)\). Since \( agr(r) \subseteq CL(\Sigma)\), then \((E' : U') \in CL(\Sigma)\) and \( \Sigma \not\models (E' : U')\). Therefore, \( \Sigma \not\models (E : U) \) otherwise \( \Sigma \models (E' : U') \) by Theorem 5.2.1.

We apply Theorem 5.3.2 to our running example.

**Example 5.6 (Constructing Armstrong Relation with anti-eUCs)** The relation \( r \) in Table 5.1 is indeed Armstrong for \( \Sigma = \{(ED : E), (EM : E)\} \) because every anti-eUC is an exact agree set of \( r \), and every exact agree set is an eUC not implied by \( \Sigma \). In fact, the anti-eUCs of \( \Sigma \) are \((E : E)\) and \((EDM : DM)\), which are the exact agree sets of the first and second tuple, and the first and third tuple, respectively. Moreover, the exact agree set of the second and third tuple is \((EDM : \emptyset)\), which is not implied by \( \Sigma \).

Given Theorem 5.3.2, it is not difficult to see that Algorithm 8 computes a relation that is Armstrong for a given set \( \Sigma \) of eUCs.

**Algorithm 8 Computing Armstrong relations**

1. **INPUT:** A set \( \Sigma \) of eUCs over a relation schema \( R \)
2. **OUTPUT:** An Armstrong relation \( r \) for \( \Sigma \)
3. Let \( t_0 \) be a tuple over \( R \) where \( t(A) = 0 \) for all \( A \in R \);
4. \( r_\Sigma \leftarrow \{t_0\} \);
5. \( i \leftarrow 1 \);
6. for each \((E : U) \in \Sigma^{-1}\) do
   7. Let \( t_i \) be a tuple over \( R \)
   8. \( t_i(A) \leftarrow \begin{cases} 0, & \text{if } A \in U \\ i, & \text{if } A \in E - U \\ \bot, & \text{if } A \in R - E \end{cases} \)
   9. \( i \leftarrow i + 1 \);
10. \( r_\Sigma \leftarrow r_\Sigma \cup \{t_i\} \);
11. **return** \( r_{Armstrong} \);

**Lemma 5.3.3 (Computing Armstrong Relation of EUC).** Algorithm 8 computes an Armstrong relation of size \( |\Sigma^{-1}| + 1 \) given a set of eUCs \( \Sigma \) over relation schema \( R \).

**Proof.** Suppose a relation \( r \) is computed by Algorithm 8 given \( \Sigma^{-1} \). For each \((E : U) \in \Sigma^{-1}\), there is \( t_i \in r \) where \( t_i(A_1) = 0, t_i(A_2) = i, t_i(A_3) = \bot \) for
all $A_1 \in U, A_2 \in E - U, A_3 \in R - E$. Hence, $t_i$ and $t_0$ exactly agree on $(E : U)$ and $\Sigma^{-1} \subseteq agr(r)$. Without loss of generality, assume there are $t_i, t_j$ where $i > j > 0$ such that $t_i, t_j$ exactly agree on $(E : U)$ and $\Sigma \models (E : U)$. According to our tuple construction, the agreed value of the two tuples is 0. So, $(E, U) \subseteq (E', U')$ where $t_0, t_i$ exactly agree on $(E' : U')$. Since $(E' : U')$ is a anti-eUC and $\Sigma \not\models (E' : U')$, $(E : U)$ also should be an anti-eUC. That is $\Sigma \not\models (E : U)$. However, it contradicts to $\Sigma \models (E : U)$ in the assumption. Therefore, $\Sigma^{-1} \subseteq agr(r) \subseteq CL(\Sigma)$. By Theorem 5.3.2, $r$ is an Armstrong relation of size $|\Sigma^{-1}| + 1$. □

**Theorem 5.3.4** (Upper Bound of EUC Armstrong Relations). For every set $\Sigma$ of eUCs, Algorithm 8 computes an Armstrong relation for $\Sigma$ with $|\Sigma^{-1}| + 1$ tuples. This number is at most the square of the minimum number of tuples required by any Armstrong relation for $\Sigma$.

**Proof.** Let $\Sigma$ be a set of eUCs over $R$. By Lemma 5.3.3, $r$ is an Armstrong relation of $\Sigma$ generated by Algorithm 8. As the construction shown in Algorithm 8, $|r| = |\Sigma^{-1}| + 1$. By Theorem 5.3.2, $\Sigma^{-1} \subseteq agr(r)$. The size of the agree set of $r$ is at most $\binom{n}{2}$, namely $|agr(r)| \leq \binom{|r|}{2}$. So, $|\Sigma^{-1}| \leq \binom{|r|}{2}$. Consequently, $|\Sigma^{-1}| \leq \frac{|r|(|r|-1)}{2}$ and $|r| \geq (1 + \sqrt{1 + 8 \cdot |\Sigma^{-1}|})/2$. So, the size $|r_{\text{min}}|$ of a minimal Armstrong relation of $\Sigma$ is $(1 + \sqrt{1 + 8 \cdot |\Sigma^{-1}|})/2$. Therefore, $|r_{\text{min}}|^2 = (1 + \sqrt{1 + 8 \cdot |\Sigma^{-1}|} + 4 \cdot |\Sigma^{-1}|) \geq |r| = |\Sigma^{-1}| + 1$. □

In practice, it may be important to focus the attention of the designers and domain experts on certain fragments of an Armstrong relation. For rows, it makes perfect sense to loop through the anti-eUCs and look at each row pair whose agree set is the anti-eUC. For columns, one may give the users of the algorithm full control over which columns should be highlighted. One sensible choice would be to inspect the columns in the embedding of an anti-eUC.
The following example demonstrates the worst case scenario in which the minimum number of tuples required by an Armstrong relation for $\Sigma$ is exponential in the input size.

**Example 5.7 (Exponential Size of Armstrong Relation)** Let $R = \{A_1, A_2, \ldots, A_{2n}\}$ where $n$ is a positive integer. Let $\Sigma = \{(A_{2i-1}A_{2i} : A_{2i-1}A_{2i}) | i = 1, \ldots, n\}$ be a set of eUCs over $R$. If $n = 2$, then

$$\Sigma^{-1} = \{(A_1A_3 : A_1A_3), (A_1A_4 : A_1A_4), (A_2A_3 : A_2A_3), (A_2A_4 : A_2A_4)\}$$

In general, $|\Sigma^{-1}| = 2^n$ where $\Sigma$ has size $4n$. 

As evidenced by Example 5.7, there is no algorithm that can compute Armstrong relations in polynomial time in the input. Extending the currently best known strategy of computing the set $\Sigma^{-1}$ of anti-keys from traditional UCs to eUCs [82], we establish a characterization of anti-keys that will provide us with an iterative algorithm to compute them.

**Lemma 5.3.5 (Computing Anti-eUCs).** Let $\Gamma = \Sigma \cup \{(E : U)\}$ be a set of eUCs over a relation schema $R$. If $(C : X) \in \Gamma^{-1}$, then the following must hold:

1. $(C : X) \in \Sigma^{-1}$, or

2. there exists $A \in U$ such that $(C : AX) \in \Sigma^{-1}$, or

3. there exists $A \in E - U$ such that $(CA : X) \in \Sigma^{-1}$, or

4. there exists $A \in E - U$ such that $(CA : AX) \in \Sigma^{-1}$.

**Proof.** Let $\Sigma \cup \{(E : U)\}$ be a set of eUCs over a relation schema $R$. $\Gamma = \Sigma \cup \{(E : U)\}$. Suppose $(C : X) \in \Gamma^{-1}$. Namely, there exists no $(E' : U') \subset (C : U)$ for any $(E' : U') \in \Gamma$. Since $\Sigma \subset \Gamma$, there also exists no $(E' : U') \subset (C : X)$ for any $(E' : U') \in \Sigma$. That is $\Sigma \not\models (C : X)$. Thus $(C : X)$ is possible to be in $\Sigma^{-1}$. If $(C : X) \in \Sigma^{-1}$, conclusion 1 trivially holds. Next, suppose $(C : X) \not\in \Sigma^{-1}$. Given $\Sigma \not\models (C : X)$, if $(C : X) \not\in \Sigma^{-1}$, then there
is \((C' : X') \in \Sigma^{-1}\) where \((C : X) \subseteq (C' : X')\). Since \((C' : X') \in \Sigma^{-1}\) and \((C : X) \in \Gamma^{-1}\), then \((E : U) \subseteq (C' : X')\) otherwise \((C', X') \in \Gamma^{-1}\) which contradicts \((C : X) \in \Gamma^{-1}\).

**Case** \(C \subseteq E\). Firstly, \(C' = C\) otherwise \((C' : X)\) must be in \(\Gamma^{-1}\) instead of \((C : X)\). Thus, \(X \subseteq X'\). Secondly, since \((E : U) \not\subseteq (C : X)\) and \(C \subseteq E\), so \(U \not\subseteq X\) and there is \(A \in U\) such that \(XA = X'\). Thirdly, \((E' : X' - Y) \not\in \Gamma^{-1}\) for all \(Y \subseteq X\) where \(|Y| \geq 2\) because \((C : X' - Y) \subseteq (C : X)\). Therefore, conclusion 2 is proved.

**Case** \(C \not\subseteq E\). Firstly, since \(C \not\supseteq E\), there is \(A \in E\) such that \(CA = C'\). Note that for any \(Y \subseteq E\) and \(|Y| \geq 2\), \((C' - Y : X) \not\in \Gamma^{-1}\) because \((C' - Y : X) \subseteq (C : X)\). Secondly, since \(C \not\supseteq E\), then \(X \subseteq X'\). On one hand, for any \(A \in U\), \((C' - \{A\} : X' - \{A\}) \subseteq (C' : X' - \{A\})\), which has been covered by the previous case. On the other hand, if there exists \(A \in E - U\) meanwhile \(A \in X'\) such that \(CA = C'\), then \((CA : XA) \in \Sigma^{-1}\). Therefore, conclusion 2 and conclusion 3 is proved.

**Lemma 5.3.5** will give us an iterative algorithm for computing the anti-eUCs for a given set of eUCs. However, in each iteration we still need to validate for each candidate anti-eUC that it is indeed an anti-eUC. This can be done efficiently with **Lemma 5.3.6**.

**Lemma 5.3.6** (Anti-eUC Test). **Validating whether a given eUC is an anti-eUC for a given set \(\Sigma\) of eUCs over relation schema \(R\) can be done in time \(O(|R| \cdot |\Sigma|)\).

**Proof.** Suppose \(\Sigma \not\models (E : U)\) where \(U \subseteq E \subseteq R\). To check if \((E : U) \in \Sigma^{-1}\), by the definition of eUCs, we have to check if \((E : U)\) always becomes a anti-eUC by extending any \(A \in R\). That is, \((E : U)\) is a anti-eUC if and only if proposition \(p : (\forall A \in R - E : \Sigma \models (EA : U)) \land (\forall A \in E - U \Sigma \models (E : UA)))\). For each \(A \in R\), making a test case only takes a constant time. Afterwards, checking if a test case can be derived from an eUC \((E, U)\) consumes time \(O(|E| + |U|)\). In total, examining all the eUCs in \(\Sigma\) consumes time \(O(|\Sigma|)\).
Therefore, the time cost for testing if \((E : U)\) is an anti-eUC is \(\mathcal{O}(|R| \cdot ||\Sigma||)\).

\[ \Box \]

Algorithm 9 iteratively examines the input eUCs in \(\Sigma\). For each input eUC \((E : U)\) it checks if any anti-eUC in \(\Sigma^{-1}\) contains \((E : U)\). The algorithm constructs \(\Gamma^{-1} = (\Sigma \cup \{(E : U)\})^{-1}\) from those anti-eUCs which belong to \(\Sigma^{-1}\) by eliminating attributes in \(U\) or \(E - U\).

**Algorithm 9 Compute Anti-eUCs**

1. INPUT: A set \(\Sigma\) of eUCs over a relation schema \(R\)
2. OUTPUT: Anti-eUCs \(\Sigma^{-1}\)
3. \(\Sigma \leftarrow \Sigma \cup \{(R : R)\}, \Sigma' \leftarrow \emptyset;\)
4. \(\Sigma^{-1} \leftarrow \{(R : R)\};\)
5. for each \((E : U)\) \(\in \Sigma\) do
6. \(\Sigma' \leftarrow \Sigma' \cup \{(E : U)\}\)
7. for each \((C : X)\) \(\in \Sigma^{-1}\) do
8. if \((E : U) \subseteq (C : X)\) then
9. \(\Sigma^{-1} \leftarrow \Sigma^{-1} \setminus \{(C : X)\}\)
10. for each \(A \in E - U\) do
11. \(\Sigma^{-1} \leftarrow \Sigma^{-1} \cup \{(C - \{A\}, X - \{A\})\}\)
12. for each \(A \in U\) do
13. \(\Sigma^{-1} \leftarrow \Sigma^{-1} \cup \{(C, X - \{A\})\}\)
14. for each \((C : X)\) \(\in \Sigma^{-1}\) do
15. if \(\exists A \in C - X \forall (E' : U') \in \Sigma' : (E' : U') \not\subseteq (C : X)\) or
16. \(\exists A \in R - U \forall (E' : U') \in \Sigma' : (E' : U') \not\subseteq (CA : X)\) then
17. \(\Sigma^{-1} \leftarrow \Sigma^{-1} \setminus \{(C : X)\}\)
18. return \(\Sigma^{-1}\)

**Theorem 5.3.7 (Computing Anti-eUC).** Algorithm 9 computes \(\Sigma^{-1}\) given a set of eUCs \(\Sigma\) over relation schema \(R\).

**Proof.** We use induction on number of inputs in \(\Sigma\) to show the loop at step 9 eventually generates \(\Sigma^{-1}\). By the construction of Algorithm 9, an input set of eUCs \(\Sigma\) is not empty. Let \(\Sigma_0 = \emptyset\) and \(\Sigma_0^{-1} = \{(R : R)\}\). On the first input \((E_1 : U_1) \in \Sigma\), obviously \((E_1 : U_1) \subseteq (R : R)\). So, \((R : R)\) cannot be a anti-eUC of \(\Sigma_1 = \Sigma_0 \cup \{(E_1 : U_1)\}\). By Lemma 5.3.5, if there is \((E : U) \in \Sigma^{-1}\), then one can find some corresponding \((E' : U') \in \Sigma_0^{-1}\). Since \((R : R)\) is the only one in \(\Sigma_0^{-1}\), it could be used to compute anti-eUCs in \(\Sigma_1\) and \((R : R) \not\in \Sigma_1^{-1}\) because \((E_1 : U_1) \subseteq (R : R)\). Let \(\Gamma_1^{-1} = \{(R - \{A\} : R - \{A\}) \mid A \in E_1 - U_1\} \cup \{(R : R - \{A\}) \mid A \in U_1\}\). By such construction, \((E : U) \in \Gamma_1^{-1}\) for all \((E, U) \in \Sigma_1^{-1}\) because \((E : U) \not\subseteq (R : R)\). In addition step 9 removes
redundancy in $\Gamma^{-1}_1$. Therefore, the non-redundant cover of $\Gamma^{-1}_1$ is equal to $\Sigma^{-1}_1$. By the statements similar to above, we can show that the non-redundant cover of $\Sigma^{-1}_i$ is equal to $\Sigma^{-1}_i$ for all $1 \leq i \leq |\Sigma|$. Therefore, Algorithm 5.3.7 is correct.

\[ y = 121.18x - 1565.3 \]
\[ R^2 = 0.9792 \]

**Figure 5.2: Average time of computing Armstrong relations**

5.4 Experiments

To evaluate the efficiency of our approach, we conducted experiments with Algorithm 8 and Algorithm 9. We randomly generated sets of eUCs over a relation schema $R$. For each set $\Sigma$ of randomly generated eUCs, we set a series of parameters: $n \in \{10, 20, \ldots, 100\}, k \in \{5, 6, \ldots, 15\}$ where $n = ||\Sigma||$ and $k = |R|$. In the experiments, we run each possible setting of the parameters 500 times and measure the average running time of the algorithm in milliseconds, the average number of tuples and null markers in the output Armstrong relation. Figure 5.2 illustrates that the average running time shows a linear growth with respect to the input size and a fixed schema. Similarly, Figure 5.3 illustrates that the size of Armstrong relations and the number of null marker occurrences grow slowly with increasing input size.
Indeed, with smaller sizes and fewer occurrences of null markers, Armstrong relations become more comprehensible to domain experts. With faster run times, communication between designers and domain experts improves in terms of frequency and efficiency.

### 5.5 Summary

In this chapter, we introduced the novel class of *embedded unique constraints* (eUCs) in the context of the relational model of incomplete data. Embedded
unique constraints target the unique identification of tuples across those tuples that meet the completeness requirements of applications. This approach ensures that the unique identification is independent of any interpretation of null markers. In order to unlock the vast usefulness of eUCs, we have studied two fundamental computational problems associated with eUCs. Firstly, we have established axiomatic and algorithmic characterizations of the implication problem, enabling us to reason efficiently about eUCs and to minimize the overhead of enforcing them within a database system. Secondly, we have further established structural and computational properties of Armstrong relations for eUCs, enabling us to represent any set of eUCs in the form of a user-friendly data sample. Our theoretical and experimental analysis shows that Armstrong relations can be computed efficiently and that their size is reasonably small in order to be effective for the acquisition of eUCs that are meaningful in a given application domain. The following chapter will investigate the discovery problem of eUCs, the computation of Armstrong samples as semantic profiles, as well as the application of eUCs to the enforcement of data integrity and the optimization of popular types of queries.
Chapter 6

Discovering Embedded Unique Constraints

In Chapter 5 we introduced the new class of embedded unique constraints (eUCs) as a robust and resource-conscious notion of unique constraints on data with missing values. In particular, Chapter 5 developed some useful tools for reasoning and acquiring eUCs at design time. However, eUCs can also provide insight into existing data. In fact, the discovery problem for eUCs facilitates a rich set of applications including data cleaning, semantic sampling, query optimization and schema design. Hence, we will use this chapter to investigate the discovery problem in detail. In practice, discovering eUCs from existing data has a much larger search space than discovering classical data dependencies such as keys or FDs. Despite these challenges, we will introduce algorithms that can discover eUCs efficiently within the likely intractability frontiers. The algorithms have been implemented and made available as part of a fully-fledged data profiling software called DataProf. The software is used to demonstrate how the discovery of eUCs can help with data cleaning and business rule acquisition in real-world scenarios. In addition, we also implement eUCs in SQL Server in the form of filtered indices. Using these implementations, we show how eUCs provide a more resource conscious notion than SQL unique constraints in maintaining data integrity and optimizing popular types of queries.

Section 6.1 provides a motivation for the discovery problem of eUCs and
Chapter 6. Discovering Embedded Unique Constraints

an overview of our results in this chapter. Section 6.2 establishes the NP-completeness and W[2]-completeness of the associated decision problem in terms of their arity. Section 6.3 introduces a new fundamental data structure on which our discovery algorithms will be based. Subsequent Section 6.4, Section 6.5, and Section 6.6 introduce the first column-efficient, row-efficient, and hybrid algorithms for the discovery of eUCs. Section 6.7 will present the analysis of our experiments with real-world benchmark data. Implementations of eUCs in SQL Server and their application in enforcing data integrity as well as in optimizing popular types of queries is examined in Section 6.8. The computation of Armstrong samples for eUCs is addressed in Section 6.9, as well as some quantitative and qualitative experiments. Section 6.10 details our data profiling software DataProf. We conclude this chapter with a brief summary in Section 6.11.

A report and demonstration of eUCs and DataProf, entitled “DataProf: Semantic profiling for iterative data cleansing and business rule acquisition”, has been published at the 2018 ACM SIGMOD Conference [106]. Section 6.8 has also been accepted for publication in the proceedings of the 35th IEEE International Conference on Data Engineering (ICDE2019) [105]. The paper is entitled “Entity Integrity, Referential Integrity, and Query Optimization with Embedded Uniqueness Constraints”.

6.1 Introduction

Keys provide efficient access to data in database systems. They are required to understand the structure and semantics of data. For a given collection of entities, a key refers to a set of column names whose values uniquely identify an entity in the collection. For example, a key for a relational table is a set of columns such that no two different rows have matching values in each of the key columns. Keys are fundamental for most data models, including semantic models, object models, XML, RDF, and graphs. They advance many
6.1. Introduction

Recent years have seen tremendous progress on the discovery of keys [3], [43], [99] despite its computational difficulty. For example, on a dataset with 50 columns, up to 126, 410, 606, 437, 752 minimal keys may exist. Indeed, deciding if some key with at most $n$ attributes holds on a given dataset is not only NP-complete, but even W[2]-complete in the size of the key. That is, the problem is likely to be intractable, even when the size of the key is fixed [20]. It is remarkable that algorithms exist that can find all minimal keys for reasonably large numbers of rows or columns [90].

Incompleteness and inconsistency impose persistently hard challenges to the discovery of data dependencies. For these reasons it is generally impossible to identify all entities uniquely. Over incomplete data, the default of discovery algorithms is to treat occurrences of the null marker $\perp$ just like ordinary domain values. That is, null is considered to be equal to null [90], [99]. This leads to outputs with a questionable semantics, since the only interpretation of $\perp$ where this makes sense is when a value does not exist. Worse, any fixed interpretation of $\perp$ is questionable, in particular in data originating from various sources. The case where null is considered to be different from null renders the validity of a key constraint independent of null marker interpretations. While SQL evaluates comparisons of $\perp$ as unknown, assigning false is consistent with the unique constraint (UC) of SQL. Indeed, a unique constraint $U$ evaluates to true on a given relation if there are no two different records that have matching non-null values on all the columns in $U$. It

<table>
<thead>
<tr>
<th>id</th>
<th>voter_id</th>
<th>name_prefix</th>
<th>first_name</th>
<th>middle_name</th>
<th>last_name</th>
<th>address</th>
<th>city</th>
<th>phone_num</th>
<th>register_date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>702</td>
<td>nell</td>
<td>mrs</td>
<td>marshall</td>
<td></td>
<td>719 carter st</td>
<td>kernersville</td>
<td></td>
<td>5/11/1940</td>
</tr>
<tr>
<td>2</td>
<td>839</td>
<td>nell</td>
<td>t</td>
<td>marshall</td>
<td></td>
<td>1731 tredgar rd</td>
<td>kernersville</td>
<td>336 992 7811</td>
<td>5/11/1940</td>
</tr>
<tr>
<td>3</td>
<td>131</td>
<td>joseph</td>
<td>t</td>
<td>cox</td>
<td></td>
<td>9 easey rd</td>
<td>new bern</td>
<td>252 000 0000</td>
<td>3/06/1935</td>
</tr>
<tr>
<td>4</td>
<td>131</td>
<td>joseph</td>
<td>thomas</td>
<td>cox</td>
<td></td>
<td>1108 highland ave #22</td>
<td>new bern</td>
<td>252 288 4763</td>
<td>3/06/1935</td>
</tr>
<tr>
<td>5</td>
<td>320</td>
<td>robert</td>
<td>f</td>
<td>boone</td>
<td></td>
<td>213 s cumberland st</td>
<td>wallace</td>
<td></td>
<td>1/01/1940</td>
</tr>
<tr>
<td>6</td>
<td>720</td>
<td>robert</td>
<td>edward</td>
<td>boone</td>
<td></td>
<td>124 rolling rd</td>
<td>burlington</td>
<td>228 8872</td>
<td>5/11/1940</td>
</tr>
<tr>
<td>7</td>
<td>962</td>
<td>margaret</td>
<td>plunk</td>
<td>isley</td>
<td></td>
<td>1860 brookwood ave #102</td>
<td>burlington</td>
<td>336 226 3774</td>
<td>10/26/1940</td>
</tr>
<tr>
<td>8</td>
<td>937</td>
<td>margaret</td>
<td>marie</td>
<td>harper</td>
<td></td>
<td>7572 bullsc rd</td>
<td>clemmons</td>
<td></td>
<td>10/26/1940</td>
</tr>
<tr>
<td>9</td>
<td>247</td>
<td>herbert</td>
<td>b</td>
<td>futrell</td>
<td></td>
<td>9802 us hwy 258</td>
<td>murfreesboro</td>
<td>252 398 3716</td>
<td>10/21/1938</td>
</tr>
<tr>
<td>10</td>
<td>244</td>
<td>sallie</td>
<td>b</td>
<td>futrell</td>
<td></td>
<td>9802 us hwy 258</td>
<td>murfreesboro</td>
<td>252 398 3716</td>
<td>10/21/1938</td>
</tr>
</tbody>
</table>

TABLE 6.1: Snippet of the ncovoter1k dataset
is surprising that SQL’s UC has not received any dedicated focus in previous studies of the discovery problem. However, even UCs are often not robust enough to accommodate peculiarities of modern day data. As an illustration, consider the data snippet of the *ncvoter1k* dataset in Table 6.1.

Here, the primary key on *voter_id* is violated. The reason may be due to an attempt to manually clean the data by assigning the same *voter_id* to both *t*₂ and *t*₃. The violation could be prevented by assigning ⊥ to the *voter_id* for either *t*₂ or *t*₃, say to *t*₂ in which information appears less reliable (see *middle_name* and *full_phone_number*). Giving up completeness, the modified relation would satisfy the UC on *voter_id*. Nevertheless, the UC on *voter_id*, being just a surrogate unique, cannot prevent the problem that the same voter may have been assigned different *voter_ids*. For example, *Nell Marshall* may refer to the same voter. Data profiling can help identify business keys that allow us to identify voters based on stable, real-life properties. A reasonable UC, denoted by *UC₁*, may be specified on the combination of *first_name*, *last_name*, and *register_date*. Indeed, a phone number or address can change and information on the *middle_name* is unreliable. Nevertheless, *UC₁* is violated in Table 6.1. The only solution that known discovery algorithms employ is to include additional columns in the constraint, such as *phone_num* and *voter_id*, resulting in the UC *UC₂* that is satisfied. The additional columns a) reduce the scope of the unique value combinations to records with no missing values on the extended combination of columns, and b) provide further opportunities for such records to be distinguishable. For example, *UC₁* has scope \{*t*₀, . . . , *t*₉\} since all tuples have no missing values on *first_name*, *last_name*, and *register_date*, while *UC₂* has only scope \{*t*₁, *t*₃, *t*₅, *t*₆, *t*₈, *t*₉\} since *t*₀, *t*₂, *t*₄, *t*₇ have missing values on *phone_num* or *voter_id*. This solution has the disadvantage that actually consistent entities, which are already distinguishable by the original set of columns, are unnecessarily subjected to additional column checks. Furthermore, any algorithms that discover all minimal UCs must keep on adding all possible column combinations until uniqueness is
achieved for all records that are complete on these column combinations. This will result in outputs with UCs that are inflated in terms of their size whenever consistent data is already unique on a smaller set of columns. Such inflation penalizes access to consistent data when the discovered constraints are used. In this chapter, we propose a different solution based on a generalization of SQL unique.

SQL unique uses the same combination of columns to stipulate completeness and uniqueness requirements. Instead, given a set $E$ of columns on which rows should be complete, it is more natural to ask which minimal subsets of columns are sufficient to identify each of these rows uniquely. More precisely, we introduced embedded unique constraints (eUCs) in Chapter 5, which consist of a pair $(E : U)$ of column combinations with $U \subseteq E$. For an eUC $(E : U)$, the combination $E$ is called the embedding, and $U$ is called the unique constraint of the eUC. Given some relation $r$, the embedding of $(E : U)$ defines the scope as the subset $r^E$ of records in $r$ which are complete on all the columns in $E$. The eUC $(E : U)$ holds on a given relation $r$ whenever the scope $r^E$ satisfies the UC on $U$. As such, UCs are the special case of eUCs where $E = U$. For example, after modifying the value 131 to $\perp$ in row $t_2$ of Table 6.1, the relation satisfies the eUC $eUC_1 = (E, U)$ where $U$ consists of first_name, last_name, and register_date, while $E$ consists of the columns in $U$ plus voter_id, phone_num. The scope of $eUC_1$ is $\{t_1, t_3, t_5, t_6, t_8, t_9\}$ and coincides with the set of rows that are complete on the columns of $UC_2$. The point is that $UC_2$ uses all five columns to identify each of the rows, while the $eUC$ is able to uniquely identify each of these rows based on the proper subset $U$ alone.

While eUCs subsume UCs as a special case, they empower users to separate completeness from uniqueness requirements. This is helpful for applications with specific completeness requirements. For example, voting campaigns may be conveniently targeted at voters that are registered with a phone number. Here, the phone numbers are not required to uniquely
identify voters, but only used to contact them. Embedded uniques also help with the integration of incomplete data. Since inclusion dependencies reference data across tables, eUCs can serve as a convenient point of reference. For example, we may want link new data to voters with phone number. For such applications, it is useful to discover the many different combinations of extensions $E$ and associated UCs $U$. It is therefore the task of data profiling to automate the process of discovering eUCs.

**Contribution.** Our main contributions in this chapter are summarized as follows.

1. We distinguish eUCs and their discovery problem from previous work.

2. We show that the decision variant of the discovery problem for eUCs is NP-complete and $W[2]$-complete in the input size.

3. We introduce a data structure called *eUC-tree* for storing and looking up eUCs efficiently.

4. We establish the first column-efficient, row-efficient, and hybrid algorithms for the discovery of eUCs. Each of these requires fundamentally new ideas to generalize and improve previous work.

5. As a special case of eUCs, we discover SQL UCs.

6. We conduct experiments on real-world benchmark datasets to demonstrate the efficiency of our algorithms, and that eUCs are effective in uniquely identifying most entities.

7. We propose the computation of Armstrong samples as a new direction in the profiling of incomplete data. Armstrong samples satisfy the same eUCs as the original dataset, but can be substantially smaller. Users may find this representation more helpful than an abstract set of constraints.

8. We conduct experiments on the computation and relative size of Armstrong samples. In particular, we compute Armstrong relations for the maximum families of eUCs, and then apply our discovery algorithms
to these relations. This generates relations with the maximum possible number of discoverable eUCs, and illustrates how quickly we can discover them.

9. We implement all the proposed algorithms into a full-fledged data profiling software called *DataProf*. We demonstrate DataProf in a real-world data cleaning application.

10. We implement eUCs of real-world incomplete data in SQL Server to enforce data integrity and optimize queries.

### 6.2 Computational Complexity

In this section, we establish the computational complexity for the decision variant of the discovery problem for eUCs. Its decision variant, $EUC$, is defined as follows.

| Problem: $EUC$ | Input: relation $r$ over schema $R$ positive integer $k$ | Output: yes, if there is some $U \subseteq E \subseteq R$ where $|E| \leq k$ and $r$ satisfies $(E : U)$ no, otherwise |

Note that $U \subseteq E$, so the cardinality $|E|$ of the embedding $E$ is an appropriate definition for the size of an eUC $(E, U)$. Our first observation is that $EUC$ is at least as hard as the decision variant $KEY$ of the key discovery problem in complete relations, defined as follows.

| Problem: $KEY$ | Input: complete relation $r$ over schema $R$ positive integer $k$ | Output: yes, if there is some $K \subseteq R$ where $|K| \leq k$ and $r$ satisfies $K$ no, otherwise |
Indeed, complete relations satisfy the key $K$ if and only if they satisfy the eUC $(K : K)$. It is known that KEY is NP-complete [13], and by reducing KEY to EUC we can establish NP-completeness for EUC, too.

**Theorem 6.2.1 (NP-completeness of EUC).** The problem EUC is NP-complete.

**Proof.** EUC is in NP, because we can guess $(E : U)$ with $|E| \leq k$ and verify in polynomial time in the input that $r$ satisfies $(E : U)$. For the NP-hardness, we reduce KEY to EUC. Take an instance $(r, k)$ of KEY where $r$ is a complete relation over relation schema $R$, and $k$ is a positive integer. Let $(r', k')$ be the instance of EUC where $r' = r$, and $k' = k$. Now it follows that a key $K \subseteq R$ with $|K| \leq k$ is satisfied by $r$ if and only if the key $K$ is satisfied by $r = r^K$ if and only if the eUC $(K : K)$ is satisfied by $r' = r$.

In recent research [20], KEY was shown to be $W[2]$-complete in the size of the key. As we can show that KEY and EUC are FPT-equivalent, it follows that the discovery of eUCs is likely to be an intractable problem even when the size of eUCs is fixed. The preliminary definitions have been stated in Section 2.5 of Chapter 2.

**Theorem 6.2.2 (Fixed-parameter Complexity).** The problem EUC is $W[2]$-complete.

**Proof.** We show that KEY and EUC are equivalent under FPT-reductions. The result then follows from the $W[2]$-completeness of KEY shown in [20].

For $\text{KEY} \leq \text{FPT} \ EUC$ we observe that the PTIME reduction in the proof of Theorem 6.2.1 is actually an FPT-reduction, since the parameter $k'$ only depends on $k$.

It remains to show that $\text{EUC} \leq \text{FPT} \ \text{KEY}$ holds as well. Let $(r, k)$ be an instance of EUC. We transform $(r, k)$ into an instance $(r', k')$ by defining $r'$ as the result of replacing null marker occurrences in $r$ with unique column values in $r'$, and defining $k'$ to be $k$. Clearly, this transformation is FPT. If there is some eUC $(E : U)$ over $R$ with $|E| \leq k$ that is satisfied by $r$, then for any two tuples $t, t' \in r'$ with $t \neq t'$ and $t[U] = t'[U]$ it would follow that
$t, t' \in r^E$ - a contradiction to $(E : U)$ being satisfied by $r$. Consequently, for any two tuples $t, t' \in r'$ with $t \neq t'$ we have $t[U] \neq t'[U]$, which means that $r'$ satisfies the key $U$ with $U \subseteq E$ and thus $|U| \leq |E| \leq k = k'$. Vice versa, if $U$ is a key with $|U| \leq k'$ that is satisfied by $r'$, then $r$ satisfies the eUC $(U : U)$ with $|U| \leq k' = k$. Indeed, if there were $t, t' \in r^U$ with $t \neq t'$ and $t[U] = t'[U]$, then $U$ would not be satisfied by $r'$. This concludes the proof.

Despite the likely intractability, even with a fixed input size, we will develop various efficient algorithms for the discovery of all minimal eUCs.

### 6.3 EUC-tree

Facing big search and solution spaces, it is of utmost importance to provide a data structure that i) can represent a minimal cover of the set of eUCs being discovered, and ii) can be used to decide whether some eUC is redundant. For this purpose, we will introduce the new data structure of eUC-trees, which we will employ in all our discovery algorithms. Our data structure generalizes the concept of antecedent trees from [41], which we recall here.

**Definition 6.1 (Antecedent Tree)** Given relation schema $R$ with a total order of attributes, an antecedent tree over $R$ is a tree such that:

1. Every node of the tree, except the root node, is an attribute of $R$, and
2. the children of a node are larger attributes w.r.t the order defined in $R$.

In an antecedent tree, attribute sets are represented as paths, and different paths of the tree represent different attribute sets. An antecedent tree can effectively store a minimal cover of a set of keys. Antecedent trees cannot represent eUCs, since the latter involve both embeddings and UCs. We therefore propose a new data structure, called eUC-trees, which serve the same purpose
for eUCs that antecedent trees serve for keys. We say that an eUC-path represents an eUC \((E : U)\) when \(E\) is the set of e-nodes of the path, and \(U\) is the set of u-nodes of the path.

**Definition 6.2 (EUC-tree)** Let \(R\) be a relation schema with a total order on its attributes. An eUC-tree is a tree with nodes that are either the root, or labeled as either e(mbedding)-nodes or u(nique)-nodes and satisfy the following properties:

1. Every node, except the root, is an attribute of \(R\);
2. All children of the root are e-nodes;
3. E-nodes can have e-node or u-node children;
4. E-node children are larger than their e-node parent;
5. U-nodes only have u-node children;
6. U-node children are larger than their u-node parent;
7. For each path of the tree from the root to a leaf, the set of u-nodes is a subset of the set of e-nodes of the path.
8. The set of eUCs, represented by the different paths from the root to the leaves of the tree, is non-redundant.

**Example 6.1 (EUC-tree)** Figure 6.1 shows an example of an eUC-tree. The tree represents a set of non-redundant eUCs including \((ABC : A)\), \((ABC : BC)\) and \((AC : C)\) over the relation schema \(\{A, B, C\}\).

Algorithm 10 decides whether a given eUC \((E : U)\) is redundant with respect to a given eUC-tree. For this we need to search for some path in the eUC-tree that represents an eUC \((E' : U')\) \(\subseteq (E : U)\). EUC-trees provide effective pruning mechanisms to support this search. The algorithm recursively traverses a chain of e-nodes and then u-nodes, starting at the root (line 23). A root node without children represents the eUC \((\emptyset : \emptyset)\), which is
subsumed by every other eUC (line 5). Whenever an e-node is visited, the next step is to recursively traverse the u-node children, and then the e-node children. The algorithm only starts traversing u-nodes if the reference of a u-node is not null (line 6). The search for a path can be limited to those with e-nodes (u-nodes, respectively) contained in \( E \) (in \( U \), respectively), see lines 19 and 15.

We will employ Algorithm 10 for the column-efficient, row-efficient, and also the hybrid discovery algorithm of eUCs, in order to check for redundancies efficiently.

### 6.4 Column-efficient Discovery

We first present a column-efficient algorithm, also known as row-based algorithm, for the discovery of eUCs.

The first step of the algorithm is to scan all pairs of distinct rows in the given relation. For each pair, we record the set \( E \) of columns on which both rows are total as well as the subset \( U \subseteq E \) of columns on which both tuples have matching values. More formally, we define the concept of non-eUC as follows.
Algorithm 10 Check Redundancies of eUCs

1: INPUT: Root node root of an eUC-tree, an eUC (E : U) over R
2: OUTPUT: true if (E : U) is redundant, false otherwise
3: function isRedundant(eNode, uNode)
4: if eNode has no children then
5: return true
6: if uNode ≠ null then
7: if uNode has no u-children then
8: return true
9: children ← the set of all u-children of uNode
10: for child ∈ children ∩ U do
11: if isRedundant(eNode, child) then
12: return true
13: for child ∈ children ∩ E do
14: if isRedundant(child, uNode) then
15: return true
16: children ← the set of all e-children of eNode
17: for child ∈ children ∩ E do
18: if isRedundant(child, uNode) then
19: return true
20: return false
21: return isRedundant(root, null) ⊳ Invoke recursion here

Definition 6.3 (Non-eUC) Let r be a relation over R and U ⊆ E ⊆ R. The pair (E : U) is called a non-eUC of r if there are distinct t_1, t_2 ∈ r such that i) t_1(A) ≠ ⊥ ≠ t_2(A) if and only if A ∈ E, and ii) t_1(A) = t_2(A) if and only if A ∈ U. A non-eUC (E : U) of r is maximal if there is no non-eUC (E' : U') of r such that (E : U) ⊏ (E' : U') holds. The set of maximal non-eUCs (MNEUs) of r is denoted by Σ^{−1}.

Note that MNEUs are essentially same as anit-eUCs discussed in Chapter 5. However, it is important to realize that MNEUs are based on a given relation but anti-eUCs are based a given set of eUCs.

The importance of Σ^{−1} for the discovery of eUCs is embodied in Theorem 6.4.1. It states that an eUC holds in a relation if and only if the eUC is not subsumed by any MNEU.

Theorem 6.4.1 (Non-eUC). Let r be a relation over R. An eUC (E : U) is satisfied by r if and only if there is no (E' : U') ∈ Σ^{−1} such that (E : U) ⊏ (E' : U').

Proof. Suppose there is some (E' : U') ∈ Σ^{−1} such that (E : U) ⊏ (E' : U').
Then there are distinct \( t_1, t_2 \in r^{E'} \) such that \( t_1(U') = t_2(U') \) holds. Since \( E \subseteq E' \) holds, \( r^{E'} \subseteq r^E \). Since also \( U \subseteq U' \) holds, there are distinct \( t_1, t_2 \in r^E \) such that \( t_1(U) = t_2(U) \) holds. Consequently, \( r \) does not satisfy \( (E : U) \).

Suppose \( r \) does not satisfy \( (E : U) \). Then there are distinct \( t_1, t_2 \in r^E \) such that \( t_1(U) = t_2(U) \) holds. Let

\[
E'' = \{ A \in R \mid t_1(A) \text{ and } t_2(A) \text{ are total} \},
\]

and

\[
U'' = \{ A \in E'' \mid t_1(A) = t_2(A) \}.
\]

It follows that \( E \subseteq E'' \) and \( U \subseteq U'' \). Hence, \( (E'' : U'') \) is a non-eUC of \( r \). Consequently, there must be some \( (E' : U') \in \Sigma^{-1} \) such that \( E \subseteq E'' \subseteq E' \) and \( U \subseteq U'' \subseteq U' \). Hence, there is some \( (E' : U') \in \Sigma^{-1} \) such that \( (E : U) \subseteq (E' : U') \) holds.

\[\Box\]

Theorem 6.4.1 forms the basis for the following iterative algorithm i.e. Algorithm 11. Here, the minimal eUCs are represented by an eUC-tree from Section 6.3. If there is no maximal non-eUC, then every eUC holds and Algorithm 11 will simply return the root node, representing the minimal cover \( \{ (\emptyset : \emptyset) \} \). Otherwise, the algorithm scans \( \Sigma^{-1} \) one by one element, and refines the current set of minimal eUCs that hold on \( r \) accordingly. Indeed, whenever a currently minimal eUC \( (E : U) \) is subsumed by the MNEU \( (M : N) \) under inspection, then the algorithm removes \( (E : U) \) in line 7 (recursively removing the leaf of the path until the current node is a non-leaf of some other path), and adds the following eUCs: for all \( A \in R - M \), \( (EA : U) \) is added, and for all \( A \in M - N \), \( (EA : UA) \) is added, unless they contain some other minimal eUC.

In addition to using an eUC-tree to minimize redundancy among the set of discovered eUCs, we further propose a heuristic which drastically improves the runtime of Algorithm 11. Particularly, we observe that the order
in which MNEUs are processed affects the runtime of Algorithm 11. In fact, some MNEUs generate more eUCs that are not in the final result than other MNEUs. For example, consider the set $\Sigma^{-1} = \{(A : A), (BC : B), (ABC : C)\}$ over $R = \{A, B, C\}$. If Algorithm 11 processes $\Sigma^{-1}$ in the order of $(A : A), (BC : B), (ABC : C)$, the generated eUCs are $(B : \emptyset), (C : \emptyset), (BC : C), (AB : \emptyset), (C : C), (AC : \emptyset), (AB : A), (AC : A), (ABC : B), (AC : AC), (BC : BC)$. If the processing order is $(ABC : C), (BC : B), (A : A)$ instead, the generated eUCs become $(A : A), (B : B), (BC : BC), (AB : B), (AC : A), (AB : A)$. The first order generates 12 eUCs but the second only 6 eUCs. Algorithm 11 prioritizes the MNEUs with larger embeddings. Applying MNEUs with smaller cardinalities later ensures that less intermediate eUCs are generated. This is because the cardinalities of embeddings and UCs keep increasing as more MNEUs are processed. Some MNEUs with smaller cardinalities will not affect any eUCs if they only generate unsatisfiable eUCs.

Algorithm 11 Column-efficient EUC Discovery Algorithm

1: **INPUT:** The set $\Sigma^{-1}$ of $r$
2: **OUTPUT:** The eUC-tree $T_\Sigma$ representing a minimal cover $\Sigma$ of those eUCs that hold on $r$
3: $T_\Sigma \leftarrow \text{root}$ \hspace{2cm} \triangleright Start with just a root node
4: for each $(M : N) \in \Sigma^{-1}$ do
5: \hspace{1cm} $\Omega \leftarrow \{(E : U) \subseteq (M : N) \mid (E : U) \text{ is an eUC-path in } T_\Sigma\}$
6: for $(E : U) \in \Omega$ do
7: \hspace{2cm} Remove eUC-path $(E : U)$ from $T_\Sigma$
8: for $A \in R - M$ do
9: \hspace{3cm} if $(EA : U)$ non-redundant or $T_\Sigma = \emptyset$ then
10: \hspace{4cm} if $T_\Sigma = \emptyset$ then
11: \hspace{5cm} $T_\Sigma \leftarrow \text{root}$
12: \hspace{4cm} Insert $(EA : U)$ as a new eUC-path into $T_\Sigma$
13: for $A \in M - N$ do
14: \hspace{3cm} if $(EA : UA)$ non-redundant or $T_\Sigma = \emptyset$ then
15: \hspace{4cm} if $T_\Sigma = \emptyset$ then
16: \hspace{5cm} $T_\Sigma \leftarrow \text{root}$
17: \hspace{4cm} Insert $(EA : UA)$ as a new eUC-path into $T_\Sigma$
18: Return $T_\Sigma$ \hspace{2cm} \triangleright If $\Sigma^{-1} = \emptyset$, then $T_\Sigma$ represents $\{(\emptyset : \emptyset)\}$

A walk-through of Algorithm 11 is demonstrated as follows.

**Example 6.2 (Column-efficient eUC Discovery Algorithm)** We illustrate how Algorithm 11 discovers the eUCs that hold in Table 6.2 over $R = \{E, D, M\}$. 

6.4. Column-efficient Discovery

<table>
<thead>
<tr>
<th>Employee</th>
<th>Department</th>
<th>Manager</th>
</tr>
</thead>
<tbody>
<tr>
<td>Homer</td>
<td>Toys</td>
<td>Burns</td>
</tr>
<tr>
<td>Homer</td>
<td>⊥</td>
<td>⊥</td>
</tr>
<tr>
<td>Marge</td>
<td>Toys</td>
<td>Burns</td>
</tr>
</tbody>
</table>

Table 6.2: An example for row-efficient eUC discovery

Comparing distinct pairs of tuples, we compute $\Sigma^{-1} = \{(E : E), (EDM : DM), (E : \emptyset)\}$. In the first iteration (starting from line 4), $\Omega = \{((\emptyset : \emptyset))\}$ since only an initial eUC $(\emptyset : \emptyset)$ is in $T_\Sigma$. For each eUC in $\Omega$ new eUCs are generated by adding extra attributes to its embedding or UC, that is eUCs $(\emptyset \cup \{D\} : \emptyset)$ and $(\emptyset \cup \{M\} : \emptyset)$ are generated since $M, D \in R - \{E\}$. In the second iteration, $\Omega = \{(D : \emptyset), (M : \emptyset)\}$. To compute new eUCs from $(D : \emptyset)$, a new eUC $(ED : E)$ is generated and added to $T_\Sigma$. From $(M : \emptyset)$, a new eUC $(EM : E)$ is generated and added to $T_\Sigma$. For the last iteration, $\Omega = \{\}$, and the algorithm terminates. The discovered eUCs from Table 6.2 are $(ED : E)$ and $(EM : E)$.  

We show Algorithm 11 works correctly.

**Theorem 6.4.2** (Column-efficient eUC Discovery). Given the set of maximal non-eUCs of a relation, Algorithm 11 computes a minimal cover of the set of eUCs that are satisfied by the relation.

**Proof.** Let $r$ be the given relation over the given relation schema $R$, and let $\Sigma^{-1}$ denote the set of maximal non-eUCs of $r$. Let $\Sigma$ denote the minimal cover of the set of eUCs that are represented by the eUC-tree $T_\Sigma$ that Algorithm 11 returns. We prove the correctness by induction over the cardinality of $\Sigma^{-1}$.

**Base case:** Here, $|\Sigma^{-1}| = 0$. So, Algorithm 11 simply returns only a root node that represents the set $\Sigma = \{((\emptyset : \emptyset))\}$. Indeed, if there are no maximal non-eUCs, then there are no non-eUCs at all, which means that every eUC is satisfied by $r$. The unique minimal cover in this case is indeed the set $\Sigma$ since $(\emptyset, \emptyset)$ is subsumed by every possible eUC. This concludes the base case.

**Inductive steps:** Here, $\Sigma^{-1} = \Gamma^{-1} \cup \{(M : N)\}$, where $\Gamma^{-1}$ denotes the set of maximal non-eUCs for some relation $r' \subseteq r$. We denote by $\Gamma$ a minimal
cover of the set of eUCs that hold on \( r' \). By induction hypothesis we know that, on input \( \Gamma^{-1} \), Algorithm 11 returns a eUC-tree \( T_\Gamma \) that represents \( \Gamma \). That is, a') \( r' \) satisfies all eUCs \((E : U)\) represented by \( T_\Gamma \), and b') if \( r' \) satisfies some eUC \((E : U)\), then there is some eUC \((E' : U') \in \Gamma \) such that \((E' : U') \subseteq (E : U)\) holds.

We will show now that a) \( r \) satisfies all eUCs \((E : U)\) represented by \( T_\Sigma \), and b) if \( r \) satisfies some eUC \((E : U)\), then there is some eUC \((E' : U') \in \Sigma \) such that \((E' : U') \subseteq (E : U)\) holds.

For a), we first look at the case where \((E : U) \in \Gamma \). Then \((E : U) \not\subseteq (M : N)\) since the eUC-path \((E : U)\) would not be represented by \( T_\Sigma \) because of line 7. Because of a') we also know that there is no \((E' : U') \in \Gamma^{-1} \) such that \((E : U) \subseteq (E' : U')\) holds. Consequently, there is no \((E' : U') \in \Sigma^{-1} = \Gamma^{-1} \cup \{(M : N)\}\) represented by \( T_\Sigma \) such that \((E, U) \subseteq (E' : U')\) holds. By Theorem 6.4.1, \( r \) satisfies \((E : U)\). We will now look at the case where \((E : U) \notin \Gamma \). Consequently, \((E : U)\) must be one of the eUCs that were added by line 12 or by line 17, so \((E : U)\) results by the augmentation of some \((E' : U') \in \Omega\) according to line 5, that is, \((E' : U') \subseteq (M, N)\). However, \((E, U)\) can therefore not be subsumed by \((M : N)\), and cannot be subsumed by any maximal non-eUCs in \( \Gamma^{-1} \). Indeed, if \((E : U)\) was subsumed by some maximal non-eUCs in \( \Gamma^{-1} \), then so would be \((E' : U')\), which would be a contradiction to \( r' \) satisfying \((E' : U')\) is in \( \Gamma \). Consequently, \((E : U)\) is not subsumed by any maximal non-eUCs in \( \Sigma^{-1} \). By Theorem 6.4.1, \( r \) satisfies \((E : U)\). This shows a).

For b), we first observe that whenever \( r \) satisfies \((E : U)\), then \((E : U) \not\subseteq (M : N)\) by Theorem 6.4.1, since \((M : N) \in \Sigma^{-1}\). Consequently, there is some attribute \( A \in E - M \) or some attribute \( A \in (U \cap E) - N \), which we denote as property (P) for easier reference later. Furthermore, whenever \( r \) satisfies \((E : U)\), then \( r' \) satisfies \((E : U)\) as well since \( r' \subseteq r \). By b'), there is some \((E' : U') \in \Gamma\) such that \((E' : U') \subseteq (E : U)\) holds. If \((E' : U') \in \Sigma\), then the proof is completed. Otherwise, \((E' : U') \in \Omega\) by line 5 and, in particular,
Now, Algorithm 11 adds new eUCs to Σ following line 12 to obtain \((E'A, U')\) for \(A \in R - M\) and line 17 to obtain \((E'A : U'A)\) for \(A \in M - N\). However, property (P) ensures that some attribute \(A\) can be picked from \(A \in E - M\) or \(A \in (U \cap E) - N\), thereby ensuring that the resulting new eUC is subsumed by \((E : U)\). Consequently, there is some \((E' : U') \in \Sigma\) such that \((E' : U') \sqsubseteq (E : U)\) holds. This proves b) and completes the proof.

6.5 Row-efficient Discovery

A row-efficient algorithm, also known as column-based algorithm, creates its search space from a given relation schema and verifies eUCs by traversing from the most general ones until all potentially valid eUCs in the search space have been examined. Attribute lattices (see Figure 2.1) have been widely used for row-efficient approaches to the discovery of data dependencies \([52], [88]\). As shown in Figure 2.1 in the preliminary (Chapter 2), level \(i\) of an attribute lattice contains all attribute sets of cardinality \(i\). In particular, attribute sets of lower levels have smaller cardinalities and represent more general uniqueness constraints (UCs). By traversing an attribute lattice from lower to higher levels, an algorithm can discover minimal UCs and prune redundant UCs in the search space. In the case of eUC discovery, the search space becomes significantly larger. For our row-efficient algorithm, we propose to use an attribute lattice, named \(u(nique\ constraint)-lattice\), to model the search space of the UCs associated with an eUC. While traversing a u-lattice, the algorithm employs another lattice, called \(e(mbedding)-lattice\) for the discovery of all minimal embeddings that apply to a given UCs. We call traversals in the u-lattice \(u-traversals\), and traversals in the e-lattice \(e-traversals\).

Our algorithms for u- and e-traversals are based on characterizations that help us validate whether a given eUC holds on the given relation. In \([52]\), the
authors proposed to use the *stripped partitions* (see Definition 2.21) of a relation to validate FDs. We will now refurnish the concept of stripped partitions so that missing values are handled properly in favor of discovering eUCs.

Let $r$ be a relation over $R$ and $U \subseteq R$. The $U$-equivalence class of tuple $t \in r$ is the set $[t]_U = \{s \in r^U \mid t[U] = s[U]\}$. The *stripped partition* of a relation $r$ over $U$ is $\pi_U(r) = \{[t]_U \mid t \in r^U, \|t[U]\| \geq 2\}$. The main use of stripped partitions in u-traversals is embodied in the following result. It provides an effective characterization to validate an eUC.

**Proposition 6.5.1 (EUC Validation).** An eUC $(E, U)$ over $R$ is satisfied by a given relation $r$ over $R$ if and only if for all $S \in \pi_U(r)$, $|r^E \cap S| \leq 1$.

**Proof.** If there is some $S \in \pi_U(r)$ such that $|r^E \cap S| \geq 2$, then there are distinct $t, t' \in r^E$ such that $t[U] = t'[U]$. This means, $r$ does not satisfy $(E : U)$.

Vice versa, if there are distinct $t, t' \in r^E$ such that $t[U] = t'[U]$, then there is some $S \in \pi_U(r)$ such that $|r^E \cap S| \geq 2$. \hfill \Box

However, the following result also shows how stripped partitions can be used in e-traversals. In fact, we can find an embedding $E$ for a given UC $U$ such that the eUC $(E : U)$ holds on $r$ if and only if each stripped partition for $U$ contains at most one total tuple. This helps us characterize effectively when we do not need to spend effort on finding an embedding for a UC.

**Proposition 6.5.2 (Existence of EUC).** Let $U \subseteq R$, and $r$ a relation over $R$. Then there is some $E \subseteq R$ with $U \subseteq E$ such that $r$ satisfies $(E : U)$ if and only if for all $S \in \pi_K(r)$, $|r^R \cap S| \leq 1$.

**Proof.** Suppose $|r^R \cap S| \leq 1$ for all $S \in \pi_U(r)$. Let $E'$ be the set $\{A \in R - U \mid \exists S \in \pi_U(r) \exists t \in S(t(A) = \perp)\}$, and let $E = E' \cup U$. Consequently, for all $S \in \pi_U(r)$, $|r^E \cap S| \leq 1$. We conclude by Proposition 6.5.1 that $r$ satisfies $(E : U)$.

Vice versa, assume that there is some $S \in \pi_U(r)$ such that $|r^R \cap S| \geq 2$. Then for all $E \subseteq R$ with $U \subseteq E$, $|r^E \cap S| \geq 2$ and $r$ does not satisfy $(E : U)$ by Proposition 6.5.1. \hfill \Box
Next, we describe the u-traversal (row-efficient algorithm) as Algorithm 12, and the e-traversal as Algorithm 13.

Algorithm 12 Unique-traversal (Row-efficient Algorithm)

1: INPUT: A relation \( r \) over relation schema \( R \)
2: OUTPUT: The eUC-tree \( T_\Sigma \) representing a minimal cover \( \Sigma \) of those eUCs that hold on \( r \)
3: \( T_\Sigma \leftarrow \emptyset \)
4: \( R' \leftarrow \{ A \in R \mid \exists t \in r \text{ such that } t(A) = \perp \} \)
5: \( \text{embs} \leftarrow \text{eTraversal}(R', \pi_\emptyset(r), \emptyset) \)
6: if \(|\text{embs}| > 0\) then
7: \( T_\Sigma \leftarrow \text{root} \)
8: for \( E \in \text{embs} \) do
9: Insert \((E : \emptyset)\) as a new eUC-path into \( T_\Sigma \)
10: \( \text{currentLevel} \leftarrow \{ A \in R \mid (A : A) \text{ non-redundant in } T_\Sigma \} \)
11: while \(|\text{currentLevel}| > 0\) do
12: \( \text{uGenNextLevel} \leftarrow \emptyset \)
13: for \( U \in \text{currentLevel} \) do
14: if \( r(U) = \emptyset \) then
15: Insert \((U : U)\) as a new eUC-path into \( T_\Sigma \)
16: continue \( \triangleright \) Goto line 13
17: \( \text{uGenNextLevel} \leftarrow \text{uGenNextLevel} \cup \{U\} \)
18: if \(|R' \cap S| \leq 1 \text{ for all } S \in \pi_U(r)\) then
19: \( R' \leftarrow \{ A \mid \exists S \in \pi_U(r), t \in S(t(A) = \perp) \} \)
20: \( \text{embs} \leftarrow \text{eTraversal}(R', \pi_U(r), U) \)
21: for \( E \in \text{embs} \) do
22: if \((E : U)\) non-redundant or \( T_\Sigma = \emptyset \) then
23: \( T_\Sigma \leftarrow \text{root} \)
24: \( \text{nextLevel} \leftarrow \emptyset \)
25: for all \( X, Y \in \text{uGenNextLevel} \) where \(|XY| = |X| + 1\) do
26: if \((XY : XY)\) is not redundant or \( T_\Sigma = \emptyset \) then
27: \( \text{nextLevel} \leftarrow \text{nextLevel} \cup \{XY\} \)
28: \( \text{currentLevel} \leftarrow \text{nextLevel} \)
29: return \( T_\Sigma \)

Algorithm 12 firstly computes the minimal embeddings for an associated UC that is empty. Subsequently, a level-wise traversal on the u-lattice starts from the singleton attribute sets (those on Level 1). On each level, those UCs that are certain to have embeddings will invoke an e-traversal as given by Algorithm 13. UCs for which no embeddings exist, and UCs with larger embeddings than themselves generate UCs for the next level. In a u-traversal, all discovered eUCs are stored in an eUC tree for fast redundancy checks. In e-traversal, instead of traversing an attribute lattice over an entire relation schema, only those attributes are used on which some tuple in some stripped partition holds a null marker. Finally, both Algorithm 12 and 13 (line 27 and 15, respectively), employ prefix blocks to generate candidates for the next
level. Prefix blocks were introduced in [7] and have been widely used for the discovery of data dependencies [3], [83], [99]. The blocks sort attribute sets in lexicographical order, and only form the union of two sets that have the same prefix on the first $k$ attributes. This ensures that all attribute sets on the next level are generated exactly once. Otherwise, candidate attribute sets on the next level need to be generated by adding one attribute at a time to attributes sets of the current level, which will result in too many redundant new candidates.

The computation of stripped partitions for UCs affects the scalability of Algorithm 12 on relations with a large number of rows. This is because it is inefficient to recompute stripped partitions for the entire relation from scratch for each UC. As another novelty, we propose Algorithm 14, which computes stripped partitions iteratively. The algorithm verifies whether tuples in the same current partition have matching total values on the new attribute. In essence, each occurring total value on the new attribute represents a new partition. Consequently, tuples of the input stripped partition are directly mapped into new partitions according to their values on the new attribute, see line 14.

**Algorithm 13 E-traversal**

```
1: INPUT: Subset $R' \subseteq R$, stripped partition $\pi_U(r)$, UC $U$
2: OUTPUT: The set $E$ of all minimal embeddings $E$ such that $(E, U)$ holds in $r$
3: $E \leftarrow \emptyset$
4: $currentLevel \leftarrow R'$
5: while $|currentLevel| > 0$ do
6:  $invalidEmbds \leftarrow \emptyset$
7:  $newValidEmbds \leftarrow \emptyset$
8:  for $E \in currentLevel$ do
9:  if $|r^E \cap S| \leq 1$ for all $S \in \pi_U(r)$ then
10:     $E \leftarrow E \cup \{U\}$
11:     $newValidEmbds \leftarrow newValidEmbds \cup \{E\}$
12:     continue
13:  $invalidEmbds \leftarrow invalidEmbds \cup \{E\}$
14:  $nextLevel \leftarrow \emptyset$
15:  for all $E, F \in invalidEmbds$ where $|EF| = |E| + 1$ do
16:     if $\forall E' \in newValidEmbds$ where $E' \subseteq EF$ then
17:        $nextLevel \leftarrow nextLevel \cup \{EF\}$
18:  $currentLevel \leftarrow nextLevel$
19: return $E$
```

In Algorithm 12, embeddings and their UCs are enumerated by cardinalities. If a UC with itself as an embedding cannot form a valid eUC, the eUCs formed by supersets of the UC may be valid and non-redundant. Such eUCs are augmented by one attribute, exhausting all possibilities. While examining a UC, all its embeddings are also enumerated by cardinalities so that only non-redundant ones are discovered. Similarly, if an embedding cannot form a valid eUC with a given UC, it is augmented by one attribute and validated on the next level. At the end, all minimal eUCs of a given relation have been computed.

Algorithm 14 Compute Stripped Partition

1: **INPUT:** Stripped partition \( \pi \) of \( r \) over \( U, A \in R \sim U \)
2: **OUTPUT:** The stripped partition \( \pi' \) of \( r \) over \( U A \)
3: \( \pi' \leftarrow \emptyset \)
4: for \( S \in \pi \) do
5: for \( t \in S \) do
6: if \( t(A) \neq \bot \) then
7: \( M[t(A)] \leftarrow M[t(A)] \cup \{ t \} \)
8: for each set \( S \) in \( M \) do
9: if \( |S| > 1 \) then
10: \( \pi' \leftarrow \pi' \cup \{ S \} \)
11: return \( \pi' \)

We show a wall-through of Algorithm 13 in the following example.

**Example 6.3 (Row-efficient eUC Discovery Algorithm)** We continue to use the relation \( r \) in Table 6.2 to demonstrate steps of discovering eUCs with the row-efficient algorithm. Note that we denote the first, second, and third tuple in Table 6.2 as \( t_1, t_2, t_3 \) respectively. First of all, there is no eUC in \( r \) whose UC is an empty set since \( \pi_\emptyset(r) = \{ r \} \) and \( |r^R \cap r| = 2 \) (Proposition 6.5.2). In other words, there is no embedding for \( \emptyset \) to form an eUC satisfied by \( r \). So, the initial set of UC candidates are \{E, D, M\}. Before starting a discovery, we pre-compute the stripped partitions for a given set of UC candidates, i.e., \( \pi_E(r) = \{ \{ t_1, t_2 \} \} \), \( \pi_D(r) = \{ \{ t_1, t_3 \} \} \), and \( \pi_M(r) = \{ \{ t_1, t_3 \} \} \). On level 1, all UC candidates need to have larger embeddings so that they may become valid eUCs (tested by line 14). Moreover, only attribute
E can have an embedding with a valid eUC (tested by line 18). At this point, Algorithm 13 will start with E and $\pi_{E}(r)$ (line 20). Here, the algorithm only takes extra attributes which are possible to form embeddings for E as embedding candidates, i.e., $\text{currentLevel} = \{M, D\}$. This way, the attribute lattice of embeddings for E becomes smaller. Algorithm 13 stops at the first iteration because M and D form minimal embeddings for E, respectively. Returning to Algorithm 12, newly discovered eUCs $(EM : E)$ and $(ED : E)$ are stored in $T_{\Sigma}$ (line 25). At the end of examining level 1, attributes D and M will be used to generate UC candidates on level 2 (line 27), i.e., DM. In addition, Algorithm 14 can be used to compute $\pi_{DM}(r)$ given $\pi_{D}(r)$ and M, i.e., $\pi_{DM}(r) = \{\{t_1, t_3\}\}$. On level 2, the only UC candidate DM cannot form any valid eUC and the row-efficient algorithm terminates. Therefore, the discovered eUCs are $(ED : E)$ and $(EM : E)$.

6.6 Hybrid Discovery

So far, our algorithms were targeted at relations with a large number of either columns or rows. Each algorithm suffers from defects that require new strategies to correct. The column-efficient algorithm has to compare all distinct rows, resulting in a quadratic growth of the runtime in the number of rows. Moreover, redundant intermediate results are produced frequently. The row-efficient algorithm operates on a huge search space, which grows exponentially in the number of columns. Since stripped partitions are created at each level of the attribute lattice, the algorithm also duplicates a lot of information, which creates problems with the available memory. As a solution, we are now proposing a hybrid algorithm that utilizes good aspects of the column-efficient algorithm to compensate defects of the row-efficient algorithm, and vice versa. This amalgamation of ideas allows us to efficiently mine datasets that have a large number of both columns and rows.

Reducing search space. The column-efficient algorithm can help reduce the number of attribute sets that both u- and e-traversals consider on each level.
Recall that non-eUCs can be used to identify invalid eUCs and to derive new satisfiable eUCs. In a u- or e-traversal, an invalid attribute set is expanded by each remaining attribute. For example, if $E$ is not an embedding for $U$, then one checks if $r$ satisfies $(EA : U)$ for all $A \in R - E$. However, if an embedding and its associated UC are subsumed by some non-eUC $(M : N)$, then one only needs to check if $r$ satisfies $(EA : U)$ for all $A \in R - M$. In fact, the row-efficient algorithm views invalid eUC as an non-eUC, and then derives new eUCs. The use of non-eUCs can thus reduce the search space in the row-efficient algorithm.

**Reducing intermediate eUCs.** The row-efficient algorithm helps the column-efficient algorithm reduce the number of eUCs generated at intermediate steps. By Theorem 6.4.1, the column-efficient algorithm cannot decide if an eUC is valid until the last MNEU has been processed. When an eUC, such as $(E : U)$, is subsumed by an non-eUC, an embedding of the eUC, such as $(EA : U)$, is either redundant or not regarding some validated eUC. If it is redundant, then all eUCs that subsume $(EA : U)$ are redundant, too. Hence, timely validation of eUCs reduces the number of intermediate eUCs generated by the column-efficient algorithm. In fact, one can validate eUCs of an eUC-tree in a level-wise manner, because levels of UCs and their embeddings can be computed by traversing the eUC-tree. For this type of pruning, we define $M_1$ and $M_2$ as mappings that assign an attribute set to some eUCs. $M_1$, called *embedding hints* (EH), is defined by $A \in M_1[E, U]$ iff $(EA : U)$ is subsumed by some valid eUC, and $M_2$, called *UC hints* (UH), is defined by $A \in M_2[E, U]$ iff $(EA : UA)$ is subsumed by some valid eUC.

Algorithm 15 updates the current eUC-tree $T_\Sigma$ based on some non-eUC $(M : N)$, EH $M_1$, UH $M_2$, and the number $l$ of levels for which eUCs with associated UC $|U| \leq l$ have been validated. The main speed-ups are achieved by line 9 and line 17, where attributes in EH and UH do no longer need to be considered when new candidate eUCs are generated for the next level. Moreover, Algorithm 15 also updates the given EH and UH for each eUC in
the eUC-tree whenever a new candidate eUC is found to be implied by some valid eUC (line 11).

Algorithm 15 Update EUCs

1: **INPUT:** eUC-tree $T_\Sigma$ over $R$, non-eUC $(M, N)$, EH $M_1$, UH $M_2$, number $l$ of validated levels
2: **OUTPUT:** An updated eUC-tree $T_\Sigma$
3: $paths \leftarrow \{\text{eUC-paths in } T_\Sigma \text{ subsumed by } (M : N)\}$
4: for each $(E : U) \in paths$ do
5: \[ P \leftarrow \emptyset \]
6: Remove eUC-path $(E : U)$ from $T_\Sigma$
7: $E \leftarrow M_1[(E : U)], U \leftarrow M_2[(E : U)]$
8: Remove $(E : U)$ from $M_1, M_2$
9: for $A \in R - (E \cup U)$ do \[
\text{▷ $E$-attributes no longer required}
\]
10: if $(EA : U) \supseteq (E' : U')$ in $T_\Sigma$ with $|U'| \leq l$ then
11: $E \leftarrow E \cup \{A\}, U \leftarrow U \cup \{A\}$
12: continue \[
\text{▷ Goto line 9}
\]
13: if $T_\Sigma = \emptyset$ then
14: $T_\Sigma \leftarrow$ a root node of an eUC-tree
15: Insert $(EA : U)$ as a new eUC-path into $T_\Sigma$
16: $P \leftarrow P \cup \{(EA : U)\}$
17: for $A \in E - (U \cup U)$ do \[
\text{▷ $U$-attributes no longer required}
\]
18: if $(EA : UA) \supseteq (E' : U')$ in $T_\Sigma$ with $|U'| \leq l$ then
19: $U \leftarrow U \cup \{A\}$
20: continue \[
\text{▷ Goto line 17}
\]
21: if $T_\Sigma = \emptyset$ then
22: $T_\Sigma \leftarrow$ a root node of an eUC-tree
23: Insert $(EA : UA)$ as a new eUC-path into $T_\Sigma$
24: $P \leftarrow P \cup \{(EA : UA)\}$
25: for each $(E' : U') \in P$ do
26: $M_1[(E' : U')] \leftarrow E, M_2[(E' : U')] \leftarrow U$

**Hybridization.** Our hybrid algorithm runs the row-efficient algorithm as its core, but employs the column-efficient algorithm to update the search space whenever convenient. This results in a *hybrid e-traversal* algorithm and a *hybrid u-traversal* algorithm.

The hybrid e-traversal validates the embeddings of a given UC level by level. Before a new level is used, hybrid e-traversal decides whether new non-eUCs should update its search space. The decision is controlled by the ratio of the number of invalid embeddings over the number of all embeddings on a level. Similar to Algorithm 13, invalid embeddings generate candidate embeddings on the next level. Hence, the more invalid embeddings are found on the current level, the more candidate embeddings need to be validated on the next level. If the ratio exceeds a certain threshold, meaning
that too many candidates would need to be validated, the search space is updated by a set of non-eUCs sampled from stripped partitions. Otherwise, the algorithm only uses non-eUCs composed by invalid embeddings to update the search space. For example, if \( E \) is not an embedding for \( U \), \((E : U)\) must be an non-eUC. Eventually, e-traversal returns updates of the eUC-tree, EHs and UHs to the u-traversal algorithm.

Unlike the u-traversal algorithm in Algorithm 12, hybrid u-traversal does not only discover the embeddings of a UC level by level, but also employs non-eUCs returned by hybrid e-traversals to update the eUC-tree at the end of each iteration. Note that hybrid e-traversals will update the entire eUC-tree, so it is no longer necessary for hybrid u-traversal to explicitly compute UCs for the next level.

We only provide a sketch to prove Theorem 6.6.1 since the main argument is similar to that of Theorem 6.4.2. The hybrid algorithms only use non-eUCs to update the eUC-tree. Non-eUCs are generated by either invalidating an eUC or by extracting samples from a stripped partition. Nevertheless, the algorithm still follows the characterization of valid eUCs by MNEUs, see Theorem 6.4.1. Our algorithms further use Proposition 6.5.1 to validate eUCs level by level. That way, EHs and UHs can be computed and employed by Algorithm 15. In Algorithm 15, whenever an eUC \((E : U)\) is subsumed by an non-eUC \((M : N)\), new eUCs will be generated regarding the non-eUC. Among the new eUCs, some of them may be implied by eUCs that have already been validated. So, for those new eUCs which are not implied but generated by the same eUC, they cannot be extended by the attributes which have been used to generate the invalid eUCs. For instance, if \((EA : U)\) is implied, but \((EB : U)\) is not implied by a set of valid eUCs, such that \(A, B \in R - M\), then an EH will map \((EB : U)\) to an attribute set containing \(A\), indicating that \((EAB : U)\) will result in an implied eUC since \((EA : U) \sqsubseteq (EAB : U)\). Lastly, since embeddings and their associated UCs
are enumerated by cardinalities, all minimal eUCs that hold on a given relation will be generated.

**Theorem 6.6.1 (Hybrid EUC Discovery).** Our hybrid discovery, which is consisted of Algorithm 15, Algorithm 16, Algorithm 17, and computes a minimal cover of the set of eUCs that are satisfied by the given relation.

### 6.6.1 Hybrid E-traversal

Algorithm 16 summarizes the techniques for our hybrid e-traversal. Based on an input UC $U$ for which it has been validated that some embedding $E$ exists for which $(E : U)$ is valid on the given input relation, and based on an antecedent tree $A_E$ that represents all candidate embeddings, Algorithm 16 validates all embeddings in $A_E$ level by level. However, the antecedent tree needs to be updated iteratively because it does not contain all the embeddings after all. Instead of using prefix blocks like Algorithm 13, the hybrid e-traversal algorithm uses invalid embeddings to construct non-eUCs, such that Algorithm 16 can update the antecedent tree with a procedure that is similar to that in Algorithm 15. Meanwhile, the input eUC-tree $T_{\Sigma}, EH$ and $UH$ are also updated when an non-eUC is processed. Note that $EH$ and $UH$ can be updated if and only if an invalid embedding is implied by embeddings on the previous levels, namely by the valid eUCs (line 20 and 30). If the ratio between the number of invalid embeddings and number of possible embeddings on the current level exceeds some threshold (0.01 found to be most suitable in our experiments), see line 11, then the invalid embeddings would generate too many candidate embeddings for the next level. In that case, the sampling of tuple pairs from our input stripped partition can generate fewer candidate embeddings (resulting in larger embedded non-uniques that subsume multiple embedded non-uniques generated by the invalid embeddings). We discuss details about our sampling method at the end of this section.
Algorithm 16 Hybrid E-traversal of EUC

1: INPUT: Associated UC $U$, eUC-tree $T_\Sigma$, antecedent tree $A_\xi$, stripped partition $\pi_U(r)$ of
2: relation $r$, EH $M_1$, UH $M_2$
3: OUTPUT: Antecedent tree $A_\xi$ with all embeddings of $U$
4: $l \leftarrow 1$
5: $\text{currentLevel} \leftarrow \{\text{paths of length } l \text{ in } A_\xi\}$
6: while the length of maximal path in $T_\Sigma$ is at least $l$ do
7: for each $E \in \text{currentLevel}$ do
8: if $|E \cap S| \leq 1$ for all $S \in \pi_U(r)$ or $r^E = \emptyset$ then
9: continue
\> Goto line 7
10: $\text{nus} \leftarrow \text{nus} \cup \{E\}$ \> Invalid embeddings
11: if $|\text{nus}|/|\text{currentLevel}| > 0.01$ then
12: Sample pairs in $S$ for all $S \in \pi_U(r)$
13: $\text{nus} \leftarrow \max(\text{nus} \cup \{\text{non-eUCs from sampling}\})$
14: for each $(M : N) \in \text{nus}$ do
15: paths $\leftarrow \{\text{paths in } A_\xi \text{ subsumed by } M\}$
16: for each $E \in \text{paths}$ do
17: Remove $E$ from $A_\xi$
18: $\mathcal{P} \leftarrow \emptyset$, $E \leftarrow M_1[(E : U)]$, $U \leftarrow M_2[(E : U)]$
19: Remove $(E : U)$ from $M_1$, $M_2$
20: for each $A \in R - (E \cup E)$ do
21: if $EA$ subsumes any $E'$ in $A_\xi$ where $|E'| < l$
22: or $(EA : U)$ subsumes any $(E' : U')$ in $T_\Sigma$
23: where $|E'| < |E|$ then
24: $E \leftarrow E \cup \{A\}$, $U \leftarrow U \cup \{A\}$
25: continue
\> Goto line 20
26: if $A_\xi = \emptyset$ then
27: $A_\xi \leftarrow \text{a root node of an antecedent tree}$
28: Insert $EA$ as a new path into $A_\xi$
29: $\mathcal{P} \leftarrow \mathcal{P} \cup \{(EA : U)\}$
30: for each $A \in E - (U \cup U)$ do
31: if $EA$ subsumes any $E'$ in $A_\xi$ where $|E'| < l$
32: or $(EA : U)$ subsumes any $(E' : U')$ in $T_\Sigma$
33: where $|U'| < |U|$ then
34: $U \leftarrow U \cup \{A\}$
35: continue
\> Goto line 30
36: if $T_\Sigma = \emptyset$ then
37: $T_\Sigma \leftarrow \text{a root node of an eUC-tree}$
38: Insert $(EA, UA)$ as a new eUC-path into $T_\Sigma$
39: for each $(E' : U') \in \mathcal{P}$ do
40: $M_1[(E' : U')] \leftarrow E$, $M_2[(E' : U')] \leftarrow U$
41: $l \leftarrow l + 1$
42: $\text{currentLevel} \leftarrow \{\text{paths of length } l \text{ in } A_\xi\}$
43: return $A_\xi$
6.6.2 Hybrid U-traversal

We now combine Algorithm 15 and Algorithm 16 to obtain Algorithm 17, which is our hybrid u-traversal algorithm. Here, we examine associated UCs level by level, traversing our eUC-tree while counting the number of visited u-nodes. If all eUCs with a fixed associated UC have been validated, they must be non-redundant because both line 18 and 21 perform redundancy checks before inserting a new eUC. Otherwise, Algorithm 16 is used to discover all minimal embeddings of the associated UC (line 15). Meanwhile, a set of non-eUCs, an EH and a UH are updated during hybrid e-traversals. Since non-eUCs acquired from hybrid e-traversals are used to update antecedent trees (Alg. 16, line 28) or the next level of the eUC-tree (Alg. 16, line 38), they also update the part of the eUC-tree which has not been validated so far.

**Algorithm 17** Hybrid U-traversal of EUC

1. **INPUT:** A relation \( r \) over relation schema \( R \)
2. **OUTPUT:** The eUC-tree \( T_\Sigma \) representing a minimal cover \( \Sigma \) of eUCs that hold on \( r \)
3. \( T_\Sigma \leftarrow \) eUC-tree with e-node \( A \) for all \( A \in R \)
4. \( M_1[(A : \emptyset)] \leftarrow \emptyset \) for all \( A \in R \) \hspace{1cm} \( \triangleright \) Initial EH
5. \( M_2[(A : \emptyset)] \leftarrow \emptyset \) for all \( A \in R \) \hspace{1cm} \( \triangleright \) Initial UH
6. \( l \leftarrow 0 \)
7. \( \text{currentLevel} \leftarrow \{ (E : U) \mid (E : U) \text{ is eUC-path with } |U| = l \} \)
8. **while** there is eUC-path \((E : U)\) in \( T_\Sigma \) where \( |U| >= l \) **do**
9. \( \text{nus} \leftarrow \emptyset \)
10. **for** each \( U \) such that \((E : U) \in \text{currentLevel} \) **do**
11. \( E \leftarrow \{ E \mid (E : U) \in \text{currentLevel} \} \)
12. **if** \( \exists E \in \mathcal{E} \) where \(|E \cap S| <= 1 \) for all \( S \in \pi_U(r) \) **then**
13. \( A_E \leftarrow \) Antecedent tree with paths in \( \mathcal{E} \)
14. **Remove** eUC-path \((E : U)\) from \( T_\Sigma \) for all \( E \in \mathcal{E} \)
15. \( E' \leftarrow \text{hybrid_Etraversal}(U, T_\Sigma, A_E, \pi_U(r), M_1, M_2) \)
16. **Add** non-eUCs used in the hybrid e-traversal to \( \text{nus} \)
17. **for** each \( E \in E' \) **do**
18. **if** \((E : U)\) non-redundant in \( T_\Sigma \) **then**
19. Insert \((E : U)\) as a new eUC-path to \( T_\Sigma \)
20. **for** each \((M : N) \in \text{nus} \) **do**
21. \( \text{updateEUCTree}(T_\Sigma, (M : N), M_1, M_2, l) \) \hspace{1cm} \( \triangleright \) Algorithm 15
22. \( l \leftarrow l + 1 \)
23. \( \text{currentLevel} \leftarrow \{ (E : U) \mid (E : U) \text{ eUC-path with } |U| = l \} \)
24. **return** \( T_\Sigma \)
6.7 Experiments

We have conducted experiments on real-world datasets to illustrate the performance and practicality of our algorithms. These datasets have emerged as benchmark datasets for testing the performance of discovery algorithms for classes such as functional dependencies [87], [88]. We implemented the proposed algorithms in Visual C++, and carried out our experiments on an Intel i7-5820K, 3.3 GHz, 8 GB, Windows 10 PC. Datasets and our software tool (DataProf) for eUCs and have been made available \(^1\).

6.7.1 Performance on Real-world Data

We first demonstrate the performance of the proposed algorithms on the benchmark datasets. For the experiments we set a time limit (TL) of 2 hours and a memory limit (ML) of 6 GB. The benchmarks include complete and incomplete datasets. For each dataset, we report the number of rows (#R), columns (#C), missing values (#⊥), incomplete rows (#IR), incomplete columns (#IC), unique constraints (#UC), eUCs (#eUC), and the runtime of each algorithm for the discovery of the eUCs. Since UCs just represent the special case of eUCs where the embedding and associated UC coincide, we have simply indicated their total number. Over complete datasets, all three notions of UCCs, UCs, and eUCs coincide. We point out that our algorithms are designed for the discovery of eUCs from incomplete data, which covers a much larger search space than the discovery problem of UCCs or UCs.

Tables 6.3 and 6.4 show our results on the incomplete and complete datasets, respectively. Since most of the incomplete datasets only have a small number of rows, the column-efficient algorithm (Algorithm 11) performs better on some of them, but has rarely a huge advantage over the hybrid algorithm. Note that neither Algorithm 11 nor Algorithm 12 can process the dataset weather [70] within the given time and memory limits. On the

\(^1\)https://bit.ly/2Xk1IIY
Chapter 6. Discovering Embedded Unique Constraints

### Table 6.3: Runtime (in seconds) of the three algorithms to discover eUCs from incomplete data

<table>
<thead>
<tr>
<th>Dataset</th>
<th>#R</th>
<th>#C</th>
<th>#⊥</th>
<th>#IR</th>
<th>#UC</th>
<th>Column-efficient</th>
<th>Row-efficient</th>
<th>Hybrid</th>
</tr>
</thead>
<tbody>
<tr>
<td>horse</td>
<td>300</td>
<td>28</td>
<td>1605</td>
<td>294</td>
<td>21</td>
<td>1.046</td>
<td>1.167</td>
<td>1.167</td>
</tr>
<tr>
<td>bridges</td>
<td>108</td>
<td>13</td>
<td>77</td>
<td>38</td>
<td>9</td>
<td>0.003</td>
<td>0.0039</td>
<td>0.002</td>
</tr>
<tr>
<td>hepatitis</td>
<td>155</td>
<td>20</td>
<td>167</td>
<td>75</td>
<td>15</td>
<td>0.082</td>
<td>17.991</td>
<td>0.154</td>
</tr>
<tr>
<td>cancer</td>
<td>691</td>
<td>11</td>
<td>16</td>
<td>16</td>
<td>1</td>
<td>0.083</td>
<td>0.187</td>
<td>0.009</td>
</tr>
<tr>
<td>echo</td>
<td>132</td>
<td>13</td>
<td>132</td>
<td>71</td>
<td>12</td>
<td>0.006</td>
<td>0.018</td>
<td>0.006</td>
</tr>
<tr>
<td>plista</td>
<td>996</td>
<td>63</td>
<td>23317</td>
<td>996</td>
<td>32</td>
<td>3.369</td>
<td>ML</td>
<td>4.177</td>
</tr>
<tr>
<td>flight</td>
<td>1000</td>
<td>109</td>
<td>51938</td>
<td>1000</td>
<td>69</td>
<td>49.367</td>
<td>ML</td>
<td>106.633</td>
</tr>
<tr>
<td>nursery</td>
<td>12960</td>
<td>9</td>
<td>1</td>
<td>34.65</td>
<td>69</td>
<td>0.346</td>
<td>1.376</td>
<td>0.067</td>
</tr>
<tr>
<td>uniprot1k</td>
<td>1000</td>
<td>223</td>
<td>179129</td>
<td>1000</td>
<td>212</td>
<td>4106.66</td>
<td>ML</td>
<td>2742.15</td>
</tr>
<tr>
<td>weather</td>
<td>262920</td>
<td>18</td>
<td>418580</td>
<td>157895</td>
<td>12</td>
<td>TL</td>
<td>ML</td>
<td>77.365</td>
</tr>
</tbody>
</table>

### Table 6.4: Runtime (in seconds) of the three algorithms to discover eUCs from complete data

<table>
<thead>
<tr>
<th>Dataset</th>
<th>#R</th>
<th>#C</th>
<th># UC</th>
<th>Column-efficient</th>
<th>Row-efficient</th>
<th>Hybrid</th>
</tr>
</thead>
<tbody>
<tr>
<td>abalone</td>
<td>4177</td>
<td>9</td>
<td>29</td>
<td>2.8</td>
<td>0.18</td>
<td>0.09</td>
</tr>
<tr>
<td>adult</td>
<td>32537</td>
<td>15</td>
<td>2</td>
<td>205.99</td>
<td>ML</td>
<td>0.64</td>
</tr>
<tr>
<td>chess</td>
<td>28056</td>
<td>7</td>
<td>1</td>
<td>116.27</td>
<td>1.25</td>
<td>0.21</td>
</tr>
<tr>
<td>iris</td>
<td>147</td>
<td>5</td>
<td>1</td>
<td>0.004</td>
<td>0.001</td>
<td>0.001</td>
</tr>
<tr>
<td>letter</td>
<td>18668</td>
<td>17</td>
<td>1</td>
<td>78.99</td>
<td>ML</td>
<td>0.55</td>
</tr>
<tr>
<td>nursery</td>
<td>12960</td>
<td>9</td>
<td>1</td>
<td>34.65</td>
<td>2.19</td>
<td>0.15</td>
</tr>
<tr>
<td>balance</td>
<td>625</td>
<td>5</td>
<td>1</td>
<td>0.06</td>
<td>0.005</td>
<td>0.004</td>
</tr>
<tr>
<td>fd</td>
<td>250000</td>
<td>30</td>
<td>3564</td>
<td>TL</td>
<td>110</td>
<td>313</td>
</tr>
</tbody>
</table>

complete datasets, the hybrid algorithm usually wins. However, the row-efficient algorithm achieves typically a better runtime on the datasets with a large number of rows. In conclusion, the hybrid algorithm performs well overall but the column- and row-efficient algorithms usually perform better on datasets with an extreme number of columns or rows. This confirms our expectations based on the design of the algorithms.

We can also discover UCCs by not choosing a symbol that is interpreted as ⊥. Their total numbers (#UCC) and corresponding runtime of our algorithms for their discovery from the incomplete datasets are shown in Table 6.5.

### 6.7.2 Scalability Test

To further analyse the row-efficiency and column-efficiency of our proposed algorithms, we analyse the discovery on projections on the dataset uniprot1k.
6.7. Experiments

<table>
<thead>
<tr>
<th>Dataset</th>
<th># UCC</th>
<th>Column-efficient</th>
<th>Row-efficient</th>
<th>Hybrid</th>
</tr>
</thead>
<tbody>
<tr>
<td>horse</td>
<td>253</td>
<td>0.283</td>
<td>ML</td>
<td>0.128</td>
</tr>
<tr>
<td>bridges</td>
<td>5</td>
<td><strong>0.003</strong></td>
<td>0.047</td>
<td><strong>0.003</strong></td>
</tr>
<tr>
<td>hepatitis</td>
<td>348</td>
<td><strong>0.06</strong></td>
<td>19.318</td>
<td>0.161</td>
</tr>
<tr>
<td>cancer</td>
<td>2</td>
<td>0.162</td>
<td>0.189</td>
<td><strong>0.009</strong></td>
</tr>
<tr>
<td>echo</td>
<td>72</td>
<td><strong>0.008</strong></td>
<td>0.026</td>
<td>0.011</td>
</tr>
<tr>
<td>plista</td>
<td>1</td>
<td>0.851</td>
<td>ML</td>
<td><strong>0.308</strong></td>
</tr>
<tr>
<td>flight</td>
<td>26652</td>
<td><strong>7.8</strong></td>
<td>ML</td>
<td>25.632</td>
</tr>
<tr>
<td>ncvoter1k</td>
<td>69</td>
<td>0.395</td>
<td>3.364</td>
<td><strong>0.051</strong></td>
</tr>
<tr>
<td>uniprot1k</td>
<td>N/a</td>
<td>ML</td>
<td>ML</td>
<td>ML</td>
</tr>
<tr>
<td>weather</td>
<td>2</td>
<td>TM</td>
<td>ML</td>
<td><strong>12.997</strong></td>
</tr>
</tbody>
</table>

Table 6.5: Runtime (in seconds) of the three algorithms to discover UCCs from incomplete data

with an increasing number of columns, and on subsets of the dataset weather with an increasing number of rows. Figure 6.2 shows how the runtime of our algorithms scales when the number of rows or columns increase, respectively. Although the row- or column-efficient algorithm perform slightly better when the number of columns or rows is small, the hybrid algorithm eventually outperforms the other two algorithms when the number of columns or rows grows larger. Again, this meets the design expectations of all algorithms: Row-/column-efficient algorithms win when there are few enough columns/rows, respectively, while the hybrid algorithm wins when column and row numbers are large enough.

6.7.3 Coverage

The more rows in the scope of an eUC, the more rows can be identified uniquely. We could use potentially different eUCs to distinguish different rows. Hence, we say that a row in a dataset \( r \) is covered if there is some discovered eUC \( (E : U) \) such that the row belongs to the scope \( r^E \). The eUC coverage of a dataset is the ratio of rows in the dataset that are covered.

Figure 6.3 shows the eUC coverage on each of the incomplete datasets. We distinguish between UCs and truly embedded UCs (where the embedding
properly contains the associated UC). Since UCs have the minimum embedding amongst all eUCs with the same associated UC, their scope has maximum cardinality. However, the point is to discover which associated UCs are sufficient for the identification of which rows. Indeed, the high number of eUCs with a high coverage of truly embedded UCs is remarkable: It shows that there are many different ways by which a large proportion of rows can already be distinguished by a proper subset of the embedding in eUCs. Recall that this was precisely our reason for studying them.

The results are even more encouraging when we look at the relative scope for each individual eUC discovered for a given dataset. This is defined as the cardinality of the scope relative to the number of rows in the given dataset. The relative scopes are shown in Figure 6.4. Evidently, there are quite a few eUCs in each dataset which can uniquely identify most of the rows, except for the dataset horse. Datasets have low coverage when several rows are ignored by eUCs, since such rows would create violations of the eUCs otherwise. In
6.7. Experiments

FIGURE 6.3: Coverage of incomplete datasets

FIGURE 6.4: Relative scope of individual eUCs
fact, rows outside the scope may represent less reliable data, because some desired eUCs would be violated if the missing information in these rows was updated.

### 6.8 Query Optimizations with EUCs

Regarding practicality of eUCs in commercial RDBMS, we now showcase the benefits of eUCs in a perspective of query optimization when they are implemented in a commercial RDBMS such as Microsoft SQL Server. We load `ncvoter1m` dataset into a database table over schema `{voter_id, voter_reg_num, name_prefix, street_address, first_name, middle_name, last_name, name_suffix, age, gender, race, ethnic, city, state, zip_code, full_phone_num, birth_place, register_date, download_month}`, which is managed by our SQL Server. Although the dataset contains 1,024,000 rows and 19 columns, most queries only take few seconds to execute since query execution plans of modern database engine wildly exploits parallelism on a server. So, measuring duration of a query is not enough to justify performance of the query, especially when the database is further expanded in the future. Hence, our performance analysis will inspect both runtime cost and estimated cost of queries. We obtain all the measures through Microsoft SQL Server Management Studio. Details of the used measures are explained as follows.

Note that estimated costs may not be accurate to approximate runtime of queries in general. However, the following examples only aim to boost plain query plans which scan an entire table with a more efficient search capability.
i.e. index seek. So, estimated cost actually gives a good measure on how well a query scales on much larger table.

6.8.1 Implementation of EUCs

EUCs can be fully implemented in modern RDBMS. On one hand, eUCs can be enforced directly into a database table by creating unique non-clustered indexes. On the other hand, eUCs can be used as separate views to reorganize physical representation of data by creating unique clustered indexes. In fact, these views are bound to the original table so they will be automatically considered by a query optimizer. Next, we give detailed examples on how to implement eUCs in SQL Server.

For simplicity, we create a table $T$ with columns $A$, $B$, $C$ in SQL server. To simply enforce eUC $ABC : C$, we can define a unique non-clustered index on $T$:

1: CREATE UNIQUE NONCLUSTERED INDEX EUC_ABC_C
2: ON T (C)
3: WHERE A IS NOT NULL AND
4: B IS NOT NULL AND
5: C IS NOT NULL

Organizing physical representation orderly is often important because non-key or non-unique columns can be easily retrieved after key look-up or index seek. To organize physical representation of data with eUCs, we can define a unique clustered index on $T$:

1: CREATE VIEW v_T
2: WITH SCHEMABINDING AS
3: SELECT A, B, C
4: FROM T
5: WHERE A IS NOT NULL AND
6: B IS NOT NULL AND
Lines 1-8 create a view from table $T$ which additionally stipulates the embedding of eUC $ABC : C$. Afterwards, a unique index can be created on the view.

Note that, with aforementioned implementations, any insertion or update that violates the enforced eUCs will raise errors in SQL Server. Also, enforced eUCs can be used to optimize a query if the query only selects rows which are in the scope of some enforced eUCs. Next, we give more detailed examples on optimizing queries with eUCs.

6.8.2 Query Optimization Examples

**Efficient Index.** As we discussed at the end of Section 6.1, UCs normally require more attributes than eUCs to uniquely identify same set of complete tuples. For instance, eUC $\{ \text{voter}_id, \text{city}, \text{register}_date, \text{first}_name, \text{last}_name, \text{age}, \text{street}_address, \text{download}_month \}$: $\{ \text{voter}_id, \text{city}, \text{register}_date, \text{download}_month \}$ is discovered from $\text{ncvoter1m}$. Let $E$ denote the embedding and $U$ the unique of the eUC. The following query can be optimized if UC $EU$ or eUC $E : U$ are enforced in the database table:

1: SELECT voter_id FROM $\text{ncvoter1m}$
2: WHERE register_date BETWEEN '1931' AND '1970' AND
3: voter_id IS NOT NULL AND city IS NOT NULL AND
4: download_month IS NOT NULL AND first_name IS NOT NULL AND
5: last_name IS NOT NULL AND street_address IS NOT NULL AND
6: age IS NOT NULL AND register_date IS NOT NULL

Line 3-6 of the query is to satisfy the completeness requirement i.e. embedding $E$ so that the database engine can safely search through a unique index
6.8. Query Optimizations with EUCs

<table>
<thead>
<tr>
<th>Index</th>
<th>CPU</th>
<th>Read</th>
<th>Write</th>
<th>Duration</th>
<th>Estimated Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>UC</td>
<td>141</td>
<td>5766</td>
<td>0</td>
<td>3080</td>
<td>4.65</td>
</tr>
<tr>
<td>EUC</td>
<td>31</td>
<td>2388</td>
<td>0</td>
<td>2964</td>
<td>2.12</td>
</tr>
</tbody>
</table>

Table 6.7: Query optimization: eUC v.s. UC

created by the UC or the eUC. When a unique index is created based on the UC or the eUC, column \textit{register\textunderscore date} is always placed at first so that this query can be optimized via index seek. SQL Server produces same query plan for the query but a plan with the index based on eUC performs better. Table 6.7 shows details of our query analysis. It is not hard to see that the index based on UC costs more on the query since it takes more time to inspect more attributes in the index.

\textbf{Efficient Join.} Building index on join conditions can also accelerate query performance. To demonstrate a join query, we create another table \texttt{ncvoter1mL} with schema \{voter\textunderscore id, city, register\textunderscore date, first\textunderscore name, last\textunderscore name, street\textunderscore address, age, download\textunderscore month\}. Corresponding rows in \texttt{ncvoter1m} table are also projected onto \texttt{ncvoter1mL}. So, eUC \{ voter\textunderscore id, city, register\textunderscore date, age, first\textunderscore name, last\textunderscore name, street\textunderscore address, download\textunderscore month \}: \{voter\textunderscore id, city, register\textunderscore date, download\textunderscore month \} is also satisfied by both tables. Similar to our previous example, the following two join queries can be optimized by enforcing the eUC and the UC as indexes respectively:

1: SELECT voter\textunderscore id FROM ncvoter1mL AS l JOIN ncvoter1m AS r
2: ON l.voter\textunderscore id = r.voter\textunderscore id AND l.city = r.city AND
3: l.register\textunderscore date = r.register\textunderscore date AND
4: l.download\textunderscore month = r.download\textunderscore month
5: WHERE l.register\textunderscore date BETWEEN '1931' AND '1970' AND
6: l.voter\textunderscore id IS NOT NULL AND l.city IS NOT NULL AND
7: l.download\textunderscore month IS NOT NULL AND l.first\textunderscore name IS NOT NULL AND
8: l.last\textunderscore name IS NOT NULL AND l.street\textunderscore address IS NOT NULL AND
9: l.age IS NOT NULL AND l.register\textunderscore date IS NOT NULL AND
10: r.voter\textunderscore id IS NOT NULL AND r.city IS NOT NULL AND
Chapter 6. Discovering Embedded Unique Constraints

<table>
<thead>
<tr>
<th>Index</th>
<th>Join Condition</th>
<th>CPU</th>
<th>Read</th>
<th>Write</th>
<th>Duration</th>
<th>Estimated Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plain</td>
<td>EUC</td>
<td>1516</td>
<td>35327</td>
<td>0</td>
<td>3426</td>
<td>32.263</td>
</tr>
<tr>
<td>Plain</td>
<td>UC</td>
<td>5542</td>
<td>35327</td>
<td>0</td>
<td>3615</td>
<td>41.369</td>
</tr>
<tr>
<td>EUC</td>
<td>EUC</td>
<td>1344</td>
<td>4815</td>
<td>0</td>
<td>3220</td>
<td>8.337</td>
</tr>
<tr>
<td>UC</td>
<td>UC</td>
<td>2696</td>
<td>12162</td>
<td>0</td>
<td>3364</td>
<td>19.329</td>
</tr>
</tbody>
</table>

Table 6.8: Query optimization: efficient join

11: r.download_month IS NOT NULL AND r.first_name IS NOT NULL AND
12: r.last_name IS NOT NULL AND l.street_address IS NOT NULL AND
13: r.age IS NOT NULL AND r.register_date IS NOT NULL

1: SELECT voter_id FROM ncvoter1m AS l JOIN ncvoter1m AS r
2: ON l.voter_id = r.voter_id AND l.city = r.city AND
3: l.register_date = r.register_date AND
4: l.download_month = r.download_month AND
5: l.first_name = r.first_name AND l.last_name = r.last_name
6: l.age = r.age AND l.street_address = r.street_address
7: WHERE l.register_date BETWEEN '1931' AND '1970' AND
8: l.voter_id IS NOT NULL AND l.city IS NOT NULL AND
9: l.download_month IS NOT NULL AND l.first_name IS NOT NULL AND
10: l.last_name IS NOT NULL AND l.street_address IS NOT NULL AND
11: l.age IS NOT NULL AND l.register_date IS NOT NULL AND
12: r.voter_id IS NOT NULL AND r.city IS NOT NULL AND
13: r.download_month IS NOT NULL AND r.first_name IS NOT NULL AND
14: r.last_name IS NOT NULL AND l.street_address IS NOT NULL AND
15: r.age IS NOT NULL AND r.register_date IS NOT NULL

Table 6.8 shows details of our analysis on the queries above and their corresponding optimizations. After indexes have been applied, the query joined by the eUC demonstrated great advantage over other queries. Note that CPU is sometimes higher than duration because CPU simply adds time costs of parallel processes together.
6.8. Query Optimizations with EUCs

### Efficient Scan

When an incomplete database table is used for reporting, one always queries complete data in order to produce more informative reports. For example, in `ncvoter1m`, one common query is to list all the voters with their addresses:

1: SELECT first_name, last_name, age, street_address, city
2: FROM ncvoter1m
3: WHERE first_name IS NOT NULL AND last_name IS NOT NULL AND age IS NOT NULL AND street_address IS NOT NULL AND city IS NOT NULL and full_phone_num IS NOT NULL

Normally, this query executes a plan to scan all the tuples in a table to check all the conditions in the `WHERE` clause. However, eUC $E: U$ is satisfied by `ncvoter1m` according to our eUC discovery where $E = \{\text{full\_phone\_num, first\_name, last\_name, age, street\_address, city} \}$ and $U = \{\text{first\_name, last\_name, age, street\_address, city} \}$. So, if the eUC is enforced to the database table, then the query will be optimized by the database engine. Table 6.9 shows details of our analysis between a plain query plan and an optimized query plan. Apparently, the optimized query plan performs better because it only has to scan the index which only contains the tuples satisfying the completeness requirement specified in the `WHERE` clause.

### Efficient Sorting

A sorted `SELECT` query is very popular in many real-world applications. For instance, in `ncvoter1m`, a common query is to list $k$ most recently retrieved voters:

1: SELECT TOP 10 voter_id, voter_reg_num FROM ncvoter1m
2: ORDER BY download_month DESC
In a plain query plan, tuples in \textit{ncvoter1m} will be sorted and then projected. However, since eUC \{\textit{voter\_id, voter\_reg\_num, city, download\_month}\}: \{\textit{voter\_id, city, download\_month}\} is satisfied by \textit{ncvoter1m} according to our eUC discovery, the database engine of SQL Server can produce a better plan if the eUC is enforced. Table 6.10 shows details of our analysis between a plain query plan and an optimized query plan. In the optimized query plan, only a short scan over the index will be sufficient because all the attributes in the query have been covered by the eUC and all the tuples have been be sorted in the index with respect to the eUC.

**Short summary.** EUCs are extremely useful for optimizing queries over incomplete databases. One one hand, ensuring uniqueness is important for indexing incomplete data because the performance of the optimized queries will not degenerate if either incomplete data is updated or new complete is inserted. On the other hand, data integrity rules represented by eUCs can be automatically assured during insertions and updates. The capability of eUCs for query optimization is not limited to our examples. In-depth studies should be further carried out. However, such advancements will heavily rely on discovering eUCs from incomplete data.

### 6.9 Armstrong Samples

Current data profiling tools return the set of constraints in a given class that hold on the given dataset. Here we propose the use of informative Armstrong samples that perfectly represent the set of discovered constraints. The Armstrong samples are subsets of the original dataset and are perfect because they satisfy the same set of constraints as the original dataset. Such
perfect samples are attractive for a number of reasons: They are accessible to a broader audience than abstract sets of constraints, and provide a good foundation for understanding why some constraints are satisfied or violated. Hence, data analysts gain insights into large datasets by inspecting the right samples. Armstrong samples are also known as informative Armstrong databases, which was studied in [34]. However, they have not been proposed as a useful tool in data profiling.

We recall the definition of an informative Armstrong sample [34]. Given a relation \( r \) and a class \( C \) of data dependencies, an Armstrong sample for \( r \) with respect to \( C \) is a subset \( r' \subseteq r \) such that \( r' \) and \( r \) satisfy the same dependencies in \( C \). Armstrong samples always exist since every relation is an Armstrong sample of itself. The idea is to find Armstrong samples of small size if possible.

As mentioned in Chapter 5, we proposed an algorithm (see Algorithm 9 and Algorithm 8) to compute Armstrong relations with synthetic data from a given set of eUCs. For computing Armstrong samples of the given dataset, we could use any of our discovery algorithms to compute \( \Sigma \) from the dataset, then compute the set \( \Sigma^{-1} \) of maximal non-eUCs from \( \Sigma \) i.e. Algorithm 9, and then choose the right tuples from the dataset to get the sample. We can also compute \( \Sigma^{-1} \) directly from the data, and then construct an Armstrong sample by picking a pair of rows from the stripped partition \( \pi_U(r^E) \) for each MNEU \( (E, U) \) in \( \Sigma^{-1} \) (see details in Algorithm 18).

Table 6.11 shows the sizes of the Armstrong samples for our benchmark datasets. Typically, the samples are much smaller than the original dataset. There are several samples whose size exceeds 20%, but this is to be expected since the size of \( \Sigma^{-1} \) can be huge, in fact similar to the number of minimal eUCs. For example, uniprot1k has only 1000 rows but more than three million eUCs are valid. Armstrong samples can be presented to end users in different ways. Instead of presenting them as one table, one could present one pair of records for each maximal non-eUC at a time.
Algorithm 18 Compute Armstrong Samples (Direct)

1: **Input:** A relation $r$ over $R$
2: **Output:** An Armstrong sample $r'$ of $r$
3: Let $r$ be an ordered set $\{t_1, t_2, \ldots, t_n\}$
4: $\Sigma^{-1} \leftarrow \emptyset$
5: $r' \leftarrow \emptyset$
6: for each $i \in \{1, \ldots, n\}$ do
7: for each $j \in \{i + 1, \ldots, n\}$ do
8: $E = \{ A \in R \mid t_i(A) \neq \bot \neq t_j(A) \}$
9: $U = \{ A \in E \mid t_i(A) = t_j(A) \}$
10: $\Sigma^{-1} = \Sigma^{-1} \cup \{(E : U)\}$
11: $\Sigma^{-1} = \{(E : U) \in \Sigma^{-1} \mid \exists (E' : U') \in \Sigma^{-1} : (E : U) \subseteq (E' : U')\}$
12: for each $(E : U) \in \Sigma^{-1}$ do
13: Select $t_1, t_2 \in S$ where $t_1 \neq t_2$ and $S \in \pi_U r$
14: $r' \leftarrow r' \cup \{t_1, t_2\}$
15: return $r'$

<table>
<thead>
<tr>
<th>Dataset</th>
<th>#R</th>
<th>#\bot</th>
<th>#IR</th>
<th>#IC</th>
<th>#MNEU</th>
<th>%Sample Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>horse</td>
<td>91</td>
<td>248</td>
<td>85</td>
<td>21</td>
<td>178</td>
<td>30.33</td>
</tr>
<tr>
<td>bridges</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>3.70</td>
</tr>
<tr>
<td>hepatitis</td>
<td>73</td>
<td>21</td>
<td>15</td>
<td>6</td>
<td>73</td>
<td>47.10</td>
</tr>
<tr>
<td>cancer</td>
<td>12</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>1.74</td>
</tr>
<tr>
<td>echo</td>
<td>28</td>
<td>19</td>
<td>13</td>
<td>5</td>
<td>16</td>
<td>21.21</td>
</tr>
<tr>
<td>plista</td>
<td>109</td>
<td>2689</td>
<td>109</td>
<td>34</td>
<td>214</td>
<td>10.94</td>
</tr>
<tr>
<td>ncvoter1k</td>
<td>114</td>
<td>255</td>
<td>114</td>
<td>5</td>
<td>65</td>
<td>11.40</td>
</tr>
<tr>
<td>flight</td>
<td>589</td>
<td>30131</td>
<td>589</td>
<td>69</td>
<td>739</td>
<td>58.90</td>
</tr>
<tr>
<td>uniprot1k</td>
<td>907</td>
<td>162048</td>
<td>907</td>
<td>212</td>
<td>14403</td>
<td>90.70</td>
</tr>
<tr>
<td>weather</td>
<td>509</td>
<td>78</td>
<td>51</td>
<td>5</td>
<td>255</td>
<td>0.19</td>
</tr>
<tr>
<td>abalone</td>
<td>56</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>30</td>
<td>1.34</td>
</tr>
<tr>
<td>adult</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>0.06</td>
</tr>
<tr>
<td>chess</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>0.04</td>
</tr>
<tr>
<td>iris</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>5.44</td>
</tr>
<tr>
<td>letter</td>
<td>32</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>16</td>
<td>0.17</td>
</tr>
<tr>
<td>nursery</td>
<td>11</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>0.08</td>
</tr>
<tr>
<td>balance</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0.80</td>
</tr>
<tr>
<td>fd</td>
<td>461</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>231</td>
<td>0.18</td>
</tr>
</tbody>
</table>

Table 6.11: Traits of real-world Armstrong samples
### 6.10 DataProf: a Semantic Profiling Software Tool for Iterative Data Cleansing and Business Rule Acquisition

#### 6.10.1 Background

Data profiling is the set of activities that determines the meta-data about a dataset. It has evolved into an area of research and enjoys tool support from...
academia and industry [2]. A strong use case is business rule acquisition to inform data cleansing, which improves the quality of the input data for analytics. This leads to better insight and decision making. Other use cases include data integration, query optimization, and data repository design [2], [60].

From a critical point of view, solutions to the discovery problem cannot actually realize the designated purpose of data profiling. The real goal is to identify business rules of the underlying application domain. These are distinguished constraints that data of the application domain is compliant with, and does not just satisfy accidentally. The separation of business rules from constraints requires knowledge and wisdom from data stewards and domain experts. However, current data profiling tools lack computational support that facilitates the interaction of these users with the data. Current tools do narrow down the search space for business rule acquisition considerably, but suffer from poor precision and recall. Here, poor precision means that many of the constraints discovered are no business rules but only satisfied accidentally. Poor recall means that actual business rules are violated due to inconsistencies, which is evidence of poor data quality. Without effective business rule acquisition, data quality cannot be defined in terms of these rules. Hence, data quality problems cannot be addressed adequately. Consequently, any analytics of poor quality data is questionable, and data-driven

| eUCs (left): (6), (0, 1, 4, 7), (2), (0, 1, 6, 7, 9), and UCCs (right): (0, 1, 4, 6, 7), (0, 1, 6, 7, 9) |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 1067444 | 2 | 1 | 1 | 1 | 2 | 1 | 2 | 1 | 1 | 2 |
| 1036172 | 2 | 1 | 1 | 1 | 2 | 1 | 2 | 1 | 1 | 2 |
| 1212422 | 4 | 1 | 1 | 1 | 2 | 1 | 3 | 1 | 1 | 2 |
| 1212422 | 3 | 1 | 1 | 1 | 2 | 1 | 3 | 1 | 1 | 2 |
| 1182404 | 3 | 1 | 1 | 1 | 1 | 2 | 1 | 2 | 1 | 1 | 2 |
| 1182404 | 3 | 1 | 1 | 1 | 1 | 2 | 1 | 2 | 1 | 1 | 2 |
| 733639 | 3 | 1 | 1 | 1 | 2 | 2 | 1 | 3 | 1 | 1 | 2 |
| 733639 | 3 | 1 | 1 | 1 | 2 | 2 | 1 | 3 | 1 | 1 | 2 |
| 734111 | 1 | 1 | 1 | 1 | 2 | 2 | 1 | 1 | 1 | 2 |
| 734111 | 1 | 1 | 3 | 2 | 3 | 1 | 1 | 1 | 2 |
| 654546 | 1 | 1 | 1 | 3 | 2 | 1 | 1 | 1 | 2 |
| 654546 | 1 | 1 | 1 | 1 | 2 | 1 | 1 | 1 | 2 |

Table 6.14: eUCs (left): (6), (0, 1, 4, 7), (2), (0, 1, 6, 7, 9), and UCCs (right): (0, 1, 4, 6, 7), (0, 1, 6, 7, 9)
decision making becomes ineffective.

For illustration consider the real-world public dataset *ncvoter1k* [88] with 1,000 records of voter information from North Carolina, USA. Each record has information about a voter including *voter_id, voter_registration_number, name_prefix, first_name, middle_name, last_name, name_suffix, age, gender, race, ethic, street_address, city, state, zip_code, phone_number, download_month*. One would expect *voter_id* to be a surrogate key, but it is not due to a duplication of the value 131. In an effort to facilitate user interaction with the data, the profiling tool should show some violation of this constraint in a sample. Here, this would show two records that differ only on their *street_address, phone_number*, and *download_month*. Non-invasive data cleansing may replace one occurrence of 131 by the null marker, namely the occurrence that is less reliable (in terms of the older record and a *phone_number* likely to be incorrect). This process is iterated until the data steward and domain expert are happy with the resulting dataset and the discovered rules.

The next generation of data profilers should have the following features to better address business rule acquisition and improve data quality. Firstly, they should facilitate user interaction with the data by iterations of constraint discovery, perfect sampling of constraint violations, and manual data cleansing on the sample that is propagated to the dataset. An important observation is that violations of actual business rules are not that frequent (‘exceptions prove the rule’). Secondly, missing information must be handled adequately to address modern applications where nulls are often used to integrate data from heterogeneous sources. In our example, replacement of multiple *voter_ids* by the null marker would violate any uniqueness constraint on *voter_id* whose semantics handles null markers just like any domain value. Thirdly, the underlying class of constraints should be expressive, in particular be able to deal with different data quality dimensions such as completeness and integrity. In our example, marketing campaigners may only be interested in uniquely identifying voters (integrity) for which a phone number
can be used to establish contact (completeness).

The additional features raise the challenge for next generation data profilers further: For example, current uniqueness constraints only address the dimensions of integrity and completeness boldly (NOT NULL means no null markers must occur at all, SQL’s unique constraint means no null marker occurrence matters); an added expressivity of the constraints may render the search space for the associated discovery problem even larger and the computational complexity harder; and it is unclear what perfect data samples constitute and how they can be computed.

In response to the motivation and challenges, we propose a demonstration of DATA PROF — the first semantic data profiler. DATA PROF implements the row-efficient, column-efficient, and hybrid discovery algorithm for the new class of embedded uniqueness constraints as well as the first algorithms for computing Armstrong samples.

6.10.2 Overview of Architecture

All algorithms discussed before form a stand-alone library which are implemented by Visual C++. These are accessible via the graphical user interface (GUI) shown in Figure 6.5, implemented in Qt. A Windows deployment can be downloaded.

User can switch among three workspaces in the GUI: Data, eUC, and A.R. (Armstrong Relation). In the Data space, CSV files can be loaded and its metadata explored. The eUC space gives access to the constraint-centric view, the A.R. space access to the data-centric view. All discovery and sampling algorithms are available for use. Using the eUC and A.R. spaces together, the users can follow the semantic profiling process of Figure 6.6. Even with no (useful) data available, the users can still start the process from either the constraint-centric view to compute an Armstrong sample, or from the data-centric view by inventing their own data sample to discover constraints. User

2https://bit.ly/2Xk1I1Y
6.10. DataProf: a Semantic Profiling Software Tool for Iterative Data Cleansing and Business Rule Acquisition

6.10.3 Demonstration Scenario

The demonstration showcases the process of Figure 6.6 on the public dataset ncvoter1k. The audience will act as domain experts while the co-authors will act as data stewards. The demo will thus mimic the real-life interactive scenario in which the tool should be used.

can freely manipulate the data and constraints to their needs. The tool can also check the validity of a given constraint on the given sample. A user guide and screen shots can be found in our deployment files.
According to Figure 6.6, all eUCs are first discovered from the given dataset. These are then looked at and questioned. Based on their professional experience with the constraints, data stewards ask questions to the domain experts for clarification. The sampling algorithms generate an Armstrong sample which is then jointly inspected by the stewards and domain experts. The smaller size of the samples gives more focus to the critical examination of the data and their underlying business rules. The tool can check the validity of any given constraint on the data. The sample helps users answer their questions, and identify inconsistencies. Any cleansing actions on the sample are propagated to the original dataset, to which the discovery algorithms are applied again. This process continues until everyone is happy or time runs out. We will now illustrate three iterations of the process to show new insights into the data and reveal likely underlying business rules.

**Iteration 1: Uncovering Surrogate Keys.** Surrogate keys can speed up data management since records can be accessed via a single integer. Looking at the dataset, one would expect that \( \text{voter}_\text{id} \) is a surrogate key: Its values are non-random sequential integers. However, the discovery of all eUCs (shown Table 6.15) from \( \text{ncvoter} \) reveals that no eUC \((\emptyset : \{\text{voter}_\text{id}\})\) with 100% coverage exists, so the surrogate key \( \text{voter}_\text{id} \) is violated. The data stewards wonder why that is. Targeted inspection of the generated Armstrong sample uncovers the two records in Table 6.16. They only differ on \( \text{address}, \text{phone}_\text{number}, \) and \( \text{download}_\text{month} \). So, the two records show us why \( \text{voter}_\text{id} \) is not a surrogate key. Indeed, the older record has less reliable information: It is unlikely to have “252 000 0000” as a phone number. What may have happened is that information of this voter was recollected but the old data was kept. In an attempt to recover the validity of the expected surrogate key, we replace the value 131 of the old record by the null marker. This does not only preserve all the information but also uncovers other meaningful eUCs. For example, when we discover eUCs based on the modified dataset, eUCs \((\emptyset : \{\text{voter}_\text{id}\})\) and \((\{\text{voter}_\text{id}\} : \{\text{registration}_\text{number}\})\) become valid. In
6.10. DataProf: a Semantic Profiling Software Tool for Iterative Data Cleansing and Business Rule Acquisition

<table>
<thead>
<tr>
<th>embedding</th>
<th>UC</th>
<th>coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>name_prefix</td>
<td>voter_id</td>
<td>2.7%</td>
</tr>
<tr>
<td></td>
<td>voter_id, middle_name</td>
<td>85.1%</td>
</tr>
<tr>
<td>name_suffix</td>
<td>voter_id</td>
<td>4.5%</td>
</tr>
<tr>
<td></td>
<td>voter_id, street_address</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>voter_id, phone_number</td>
<td>38.8%</td>
</tr>
<tr>
<td></td>
<td>voter_id, download_month</td>
<td>100%</td>
</tr>
</tbody>
</table>

Table 6.15: Embedded uniqueness constraints using voter_id

<table>
<thead>
<tr>
<th>voter_id</th>
<th>street_address</th>
<th>phone_number</th>
<th>download_month</th>
</tr>
</thead>
<tbody>
<tr>
<td>131</td>
<td>1108 highland ave #22</td>
<td>252 288 4763</td>
<td>2011-12</td>
</tr>
<tr>
<td>131</td>
<td>9 Casey rd</td>
<td>252 000 0000</td>
<td>2011-10</td>
</tr>
</tbody>
</table>

Table 6.16: Snapshot of Armstrong sample for ncvoter

Particular, old and new record are still linked via the same value on registration_number.

Iteration 2: Uncovering Futile Attributes. Further inspection of the discovered eUCs, many contain the attributes name_prefix or name_suffix in their embedding $E$. However, the coverage of these eUCs is only approximately 2.7%. As shown in Figure 6.7, most values of these attributes are missing. For existing values such as “Jr”, “Sr”, or "Mrs", only limited information becomes available. Indeed, all voters with an existing value on name_suffix can already be distinguished by first_name and last_name. In addition, no eUC contains name_prefix or name_suffix in their set $U$ of attributes. One may at least strongly consider the removal of these attributes from the dataset.

Figure 6.7: Values of name_prefix (left) and name_suffix (right)
Iteration 3: Uncovering Implicit Missing Data. Continuing from the previous iteration, one interesting minimal eUC draws our attention after another round of discovery:

\[
(\{\text{middle\_name}\} : \{\text{first\_name}, \text{zip\_code}, \text{phone\_number}\})
\]

means there are two different records with matching values on \text{first\_name}, \text{zip\_code}, and \text{phone\_number}, and one of them should not have a middle name. This eUC indicates an highly unusual scenario in the real-world. Indeed, it is unusual for different people with the same phone number to have the same first name. In particular, the previous iteration showed that situations like James Bond Sr and James Bond Jr do not occur. After re-sampling the data, we found the two records shown in Table 6.17. These voters have the same first name and live in the same area, but both have phone number “000 0000”. In fact, 51 out of 388 voters have the phone number “000 0000”. This strongly suggests to replace the occurrences of this value by the null marker. In fact, doing so will result in the validity of the business rule \((\emptyset : \{\text{first\_name}, \text{zip\_code}, \text{phone\_number}\})\).

<table>
<thead>
<tr>
<th>voter_id</th>
<th>first_name</th>
<th>middle_name</th>
<th>zip_code</th>
<th>phone_number</th>
</tr>
</thead>
<tbody>
<tr>
<td>687</td>
<td>Margaret</td>
<td>Caudle</td>
<td>27215</td>
<td>000 0000</td>
</tr>
<tr>
<td>952</td>
<td>Margaret</td>
<td>⊥</td>
<td>27215</td>
<td>000 0000</td>
</tr>
</tbody>
</table>

Table 6.17: Sampled records with suspicious phone numbers

6.11 Summary

We presented the first discovery algorithms for \textit{embedded unique constraints} (eUCs). These include known unique constraints, such as SQL’s unique constraints or unique column combinations, as simple special cases. We showed that the problem of discovering an eUC with a given maximum size is both NP-complete and \(W[2]\)-complete in the input size. Despite the computational challenge, we established the first column-efficient, row-efficient, and
hybrid algorithms for the discovery of all eUCs that hold on a given relation. Our hybrid algorithm is particularly suited for datasets with large number of columns and rows. Our experiments confirm that the many eUCs discovered, in particular those with higher relative scope, offer opportunities for targeted and fast access to data by applications with completeness and uniqueness requirements. There were many non-trivial eUCs that can uniquely identify every row in most of the datasets we analysed. In addition, we proposed Armstrong sampling as a new direction in data profiling. Finally, we implemented a fully-fledged software tool to showcase the first semantic data profiler, DataProf. DataProf essentially organizes all the algorithms that discover eUCs and compute Armstrong samples together in a user-friendly GUI. With the support of DataProf, we introduced a novel iterative discovery and sampling process that help experts interact with data and data profiles in order to improve data quality and acquire meaningful business rules. Such process was exemplified in a real-world scenario by DataProf.
Chapter 7

Data Completeness Tailored

Database Design

In this chapter, we will focus on tailoring schema design to the completeness requirements of applications. The motivation for this work and an overview of our results are detailed in Section 7.1. Indeed, the well-known framework of relational schema design only addresses the integrity dimension of data quality, and it is surprising that other data quality dimensions have not been added by previous work. In Section 7.2 we outline our wider vision for lifting data quality dimensions to first-class citizens of schema design, and exemplify our ideas. In Section 7.3 we introduce the novel class of embedded functional dependencies as an expressive class that is robust under different interpretations of null markers, combines the specification of integrity and completeness requirements, provide a rich source for data redundancy, and facilitate the development of efficient solutions to important computational problems. In the same section we establish axiomatic, algorithmic, and logical foundations for the implication problem associated with the combined class of embedded unique constraints and embedded functional dependencies. In Section 7.4, these foundations will allow us to establish generalizations of Boyce-Codd and Third Normal Forms that do not permit any redundancy in any future application data, or minimize their redundancy across dependency-preserving decompositions, respectively. In Section 7.5 we will show how to transform any given schema into application schemata.
that meet given data quality requirements and the conditions of the general-
ized normal forms. Data over those application schemata are therefore fit
for purpose by design. We will employ extensive experiments with bench-
mark schemata and data in Section 7.6 to illustrate our framework, and the
effectiveness and efficiency of our algorithms, but also provide quantified
insight into database schema design trade-offs. We conclude this chapter in
Section 7.7 with a brief summary.

7.1 Introduction

The primary aim of database design is to find a schema that facilitates
the processing of the future application workload as best as possible. The
well-known database design framework for relational databases is centered
around the notion of data redundancy [18], [77], [94]. In practice, the major-
ity of redundant data value occurrences originate from functional dependen-
cies (FDs) [19]. These dependencies encode important integrity requirements
of the underlying application domain. Classical database design provides a
framework to compute schemata in which the integrity requirements can be
enforced efficiently during updates. Surprisingly, database research has not
brought forward schema design frameworks that address other important
application requirements. This is surprising for traditional database applica-
tions already, but even more so in a modern context. In fact, any applications
require data to be fit for purpose [96], [97]. The cost of insufficient data qual-
ity is enormous from multiple angles [92]: A Gartner survey estimates that
“poor-quality data is costing organizations on average USD 14.2 million an-
nually”. Ovum Research reported that poor data quality data is costing busi-
nesses at least 30% of revenues. Forrester reports that “Nearly one third of
analysts spend more than 40 percent of their time vetting and validating their
analytics data before it can be used for strategic decision-making”. Produc-
tivity is impacted as data scientists spend from 50 to 80 percent of their time
on collecting and preparing unruly digital data, before it can be explored for useful nuggets. An important observation is that different application requirements will best be met by different schema designs. However, relational schema design only considers the integrity requirements of data, but not other data quality dimensions. One of the most important data quality dimensions in many application areas is data completeness [96]. For example, data completeness is seen as a key to customer satisfaction [21], patient recruitment for clinical trials [64], credit risk management [85], open government data [103], and disaster prevention [40]. Value completeness is very fundamental, as missing data values occur already frequently in traditional applications, but even more so in modern applications such as data integration [35], [56]. The quality of analytics and data science projects can only be as good as the quality of the input data. Unfortunately, already the processing of data with missing values is still not well understood. In query processing, even the formulation of a meaningful answer to queries over incomplete data is challenging [5]. The few approaches to schema design for data with missing values all depend on the interpretation of null markers that represent missing data [60], [67].

What is still absent, is a generic framework to schema design based on the data quality requirements of applications. Our overall aim is to address this shortcoming and incorporate data quality requirements within business rules that are a major source of data redundancy. Firstly, we would like to enable data stewards to declare data quality requirements of applications as part of familiar types of business rules. Secondly, we would like to develop a new schema design framework that empowers data stewards to compute schemata that are fit for application purpose by design. Thirdly, the new framework should be simple yet general enough for use in practice, in particular extend the relational framework naturally.

Our contribution in this chapter is summarized as follows.

1. We propose a conceptual framework for data-quality driven database
schema design, using applications requirements to generate schemata that are fit with purpose.

2. We propose a new class of embedded functional dependencies (eFDs) that generalize FDs by incorporating data completeness requirements.

3. We develop a full design theory including axiomatic problem for eFDs and eUCs.

4. We define a schema to be in $E$-Redundancy Free Normal Form (E-RFNF) if every instance over the schema does not exhibit any $E$-redundant data value occurrences. We characterize $E$-RFNF syntactically by an extension of the well-known Boyce-Codd Normal Form (BCNF), called $E$-BCNF.

5. We provide a suitable extension of the well-known Third Normal Form (3NF), called $E$-3NF.

6. We show how to transform any given set of eUCs and eFDs into a set of classical FDs such that classical lossless BCNF-decomposition and lossless 3NF-synthesis algorithms results in schemata that are in $E$-BCNF and $E$-3NF, respectively.

7. We conduct comprehensive experiments on real-world benchmark data and schemata that show effectiveness and efficiency of our proposed framework, but also quantify insights on data redundancy and classical normalization in unprecedented form.

7.2 Vision and Example

Before getting into technical details, we illustrate the conceptual ideas of our design framework with a real-world example.

The conceptual idea is summarized in Figure 7.1. Motivated by modern day practice, data is collected from a variety of sources such as data markets, multimedia gadgets, data warehouses, social media, or public data sources.
The raw data is then stored in data lake. The raw data undergoes a cleaning process to get it ready for use in applications. We refer to the outcome of the data preparation and cleaning process as master data. The master data is augmented by meta-data about its data’s quality dimensions, such as accuracy, completeness, integrity, or timeliness [96]. The core idea of our framework is to tailor database schema design to applications. This is done by considering only those data that meet the quality requirements of the application. In Figure 7.1, the checklist icons illustrate the different data quality requirements of the different applications on the right-hand side. As illustrated, each list of requirements drives the transformation of the master data schema into database schemata for any application with this list of requirements. Hence, the resulting application schema is fit for purpose by design, and data that meets the application requirements can be processed efficiently.

A special case of this framework is classical relational database schema design. Here, data integrity is the only data quality dimension that is supported. In fact, integrity requirements are represented by integrity constraints, most commonly functional dependencies (FDs). Functional dependencies provide declarative means to encode business rules and capture a
major source of data redundancy. Data redundancy is good for processing queries but bad for processing updates. For this reason, functional dependencies are instrumental for classical database schema design with normalized schemata in Boyce-Codd Normal Form (BCNF) or Third Normal Form (3NF). Our vision is to identify more general classes of constraints that also encode other data quality dimensions in a way that enables us to tailor schema design to data meeting these requirements.

Recall that an FD is of the form $X \rightarrow Y$ with attribute subsets $X, Y$, expressing that every pair of records with matching values on all the attributes in $X$ must also have matching values on all the attributes in $Y$. Hence, given any two different records with matching values on all the attributes in $X$, the data value occurrences on each of the attributes in $Y$ for one of these records are redundant since they coincide with those of the other record. Redundant data value occurrences can be eliminated from any future database instance by ensuring that for each FD $X \rightarrow Y$ the LHS $X$ forms a key for the underlying schema, that is, no two different records have matching values on all the attributes in $X$. Classical FDs and normalization do not consider data with missing values. Our idea is to extend FDs to a new class of integrity constraints called embedded functional dependencies (eFD) by incorporating data completeness requirements. An eFD is an expression $E : X \rightarrow Y$ where $X, Y \subseteq E$ are all attribute subsets. Here, $E$ is called the embedding of the eFD and encodes the completeness requirements by saying that the classical FD $X \rightarrow Y$ must hold on the collection of records that have no missing data values on any of the attributes in $E$. In other words, the eFD $E : X \rightarrow Y$ is satisfied by a given relation $r$ whenever the classical FD $X \rightarrow Y$ is satisfied by the relation $r^E \subseteq r$ that consists of all the records in $r$ that are complete on $E$. Similar to how FDs cause data redundancy, eFDs cause $E$-redundant data value occurrences, that is redundant data value occurrences in records that are $E$-complete. Our work will show that $E$-redundant data value occurrences can be eliminated from any future database instance by ensuring that
for each eFD $E : X \rightarrow Y$, the expression $E : X$ forms an *embedded unique constraint* (eUC) for the underlying schema, that is, no two different $E$-complete records have matching values on all the attributes in $X$.

For illustration purposes on a real-world example, let us consider the benchmark dataset *ncvoter1k* that provides various information about voters. It consists of 19 columns and 1,000 records. We have mined *ncvoter1k* with respect to the eFDs and eUCs that it satisfies. Some of these mined eFDs and eUCs may only hold accidentally in this dataset, while other eFDs or eUCs may encode meaningful business rules for the voting domain but may be violated due to dirty data. As we are not domain experts, we simply make the assumption that the dataset satisfies exactly all meaningful eFDs and eUCs. This is sufficient for illustration purposes as we do not aim at discussing data cleaning nor how to acquire application requirements. Based on this assumption, the left-hand side of Figure 7.2 shows part of the data quality profile for the dataset, including the number of mined eFDs and eUCs, the number of records that are $E_1$- or $E_2$-complete, and data value occurrences that are $E_1$- or $E_2$-redundant, where $E_1$ and $E_2$ are the attribute sets that encode the data completeness requirements of two different applications. In this example, the first application is concerned with records that have actual contact details about voters, while the second application requires additional personal details about the voters. For instance, the eFD

---

**Figure 7.2:** Real-world example of data-completeness tailored database schema design
Chapter 7. Data Completeness Tailored Database Design

$E_1 : \text{full\_phone\_num} \rightarrow \text{last\_name}$ causes 12 distinct $E_1$-redundant data value occurrences, with each of the following projections occurring for two voters.

<table>
<thead>
<tr>
<th>full_phone_num</th>
<th>last_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>704 272 8433</td>
<td>horne</td>
</tr>
<tr>
<td>336 993 3583</td>
<td>idol</td>
</tr>
<tr>
<td>226 1717</td>
<td>thompson</td>
</tr>
<tr>
<td>252 398 3716</td>
<td>futrell</td>
</tr>
<tr>
<td>252 569 1361</td>
<td>newman</td>
</tr>
<tr>
<td>252 232 2597</td>
<td>etheridge</td>
</tr>
</tbody>
</table>

The right-hand side of Figure 7.2 shows characteristics of the schema designs that have been computed with respect to the eFDs and eUCs that meet the data-completeness requirements of the two different applications. This chapter develops the entire foundations, normal forms, and normalization that make such computations possible. As we see, there are extensions of BCNF and 3NF tailored to the completeness requirements, and each addressing different trade-offs. On one hand, $E$-BCNF eliminates all $E$-redundant data value occurrences, but may not preserve all given eFDs. On the other hand, $E$-3NF preserves all given eFDs but at the cost of not being able to eliminate all $E$-redundant data value occurrences. This is further illustrated in our example by showing actual statistics about the levels of preservation and redundancy elimination. In fact, there is some actual data about the projections of ncvoter1k to the normalized schemata. Here, 336 records meet the completeness requirements of the first application, while 300 records meet those of the second application. The $E$-3NF of the first application eliminates more than half of the redundant data values. However, the $E$-3NF of the second application actually exhibits more redundant data values, since some of them have been duplicated over multiple schemata. Indeed, the eFD $\text{full\_phone\_num, birth\_place : zip\_code} \rightarrow \text{city}$ causes 235 $E_2$-redundant data value occurrences on the original dataset. As zip\_code $\rightarrow$ city holds on two
of the normalized schemata that are in 3NF but no in BCNF, this FD alone causes 470 redundant occurrences in the projection of \textit{ncvoter1k} onto the 3NF decomposition. For instance, the value 'Burlington' on \textit{city} occurs for 34 different voters with complete values on \textit{full_phone_num} and \textit{birth_place}, each with the \textit{zip_code} '27215'.

\section*{7.3 Foundations}

\subsection*{7.3.1 Basics}

Our approach to capturing data semantics for incomplete relations is data-centric. First of all, we let the application decide which data completeness requirements tuples must meet to be fit for use by the application. That is, we embed the data completeness requirements in the declaration of constraints. Secondly, the semantics of our constraints is based exclusively on the complete information embedded in the underlying relation. In other words, we follow the principled approach that missing values in the form of null marker occurrences must not impact the decision whether a constraint is satisfied by the given relation or not. This decision is entirely determined by the actual data values that are available.

In Chapter 5, we have introduced \textit{embedded unique constraint} (eUCs) in Definition 5.1. Given a relation schema \( R \), an \textit{embedded unique constraint} (eUC) is an expression of the form \( E : U \) where \( U \subseteq E \subseteq R \). A relation \( r \) satisfies the eUC \( E : U \) if and only if the scope \( r^E \) satisfies the unique constraint \( U \). In the special case where \( E = U \), the eUC \( U : U \) is satisfied by a relation \( r \) if and only if the unique constraint \( U \) is satisfied by \( r \) using the \( \perp \neq \perp \) semantics if and only if the SQL unique constraint on \( U \) is satisfied. Of course, if \( E \) contains some attribute that is not in \( U \), then the semantics of eUCs cannot be captured by any other notion of a key, as they do not separate the completeness and uniqueness requirements. The decision to require \( U \subseteq E \) ensures that the semantics of the eUC only depends on the complete fragments embedded
in the underlying relation. Based on this, we will now formally introduce embedded functional dependencies.

**Definition 7.1 (Embedded Functional Dependency)**  Given a relation schema $R$, an embedded functional dependency (eFD) is an expression of the form $E : X \rightarrow Y$ where $XY \subseteq E \subseteq R$. A relation $r$ satisfies the eFD $E : X \rightarrow Y$ if and only if the scope $r^E$ satisfies the functional dependency $X \rightarrow Y$.

Given $E : X$ or $E : X \rightarrow Y$, we sometimes simply write $E - X : X$ or $E - (XY) : X \rightarrow Y$, respectively, to emphasize which additional attributes apart from those in $XY$ are required to have no missing values. The following running example is motivated from the `ncvoter1k` dataset, but compressed to make an illustration throughout this chapter. We consider the attributes $f(\text{first_name}), l(\text{last_name}), p(\text{hone}), \text{and } d(\text{ate_register})$, representing information about the first and last name of voters, their preferred phone number, and the date they registered for the vote. Let $r$ refer to the relation in Table 7.1.

Here, the key $\{f, p\}$ is violated under both $\bot = \bot$ and $\bot \neq \bot$ semantics, and so is the eUC $fp : fp$. However, the eUC $dfp : fp$ is satisfied since the score $r^{dfp}$ only consists of the first two tuples, which have different $f\_names$. Under $\bot = \bot$ semantics, the FD $d \rightarrow l$ is violated, but under $\bot \neq \bot$ semantics, it is satisfied. In either semantics, the FDs $p \rightarrow d$ and $p \rightarrow l$ are violated. The eFD $pl : p \rightarrow l$ is also violated, while the eFD $dpl : p \rightarrow l$ is satisfied. The two eFDs illustrate how the semantics of an FD changes under different data completeness requirements: The phone number identifies the last name...
of voters when information about their registration date is known, but that is not true when this information is missing.

Every total relation over $R$ satisfies the FD $X \rightarrow R$ if and only if it satisfies the key $X$. This relationship occurs in our framework as well: A relation over $R$ satisfies the eFD $R : X \rightarrow R$ if and only if it satisfies the eUC $R : X$. Indeed, if a relation satisfies $E : X$, then it also satisfies $E : X \rightarrow E$, but not necessarily vice versa. In fact, the relation in Table 7.1 satisfies the eFD $dpl : dp \rightarrow dpl$, but it violates the eUC $dpl : dp$. This observation is important, as it does not suffice for our targeted schema design framework to consider eFDs in isolation from eUCs. In particular, eFDs invite data redundancy, while eUCs refuse data redundancy. Hence, the combined class of eFDs and eUCs needs to be studied. This is different from the familiar special case of total relations where any key $X$ over $R$ (an eUC of type $R : X$) can be expressed by the FD $X \rightarrow R$ (an eFD of type $R : X \rightarrow R$).

Continuing with our example, we regard the eFD $d : p \rightarrow l$ as a meaningful constraint of our application domain. Then it indicates that there are relations that exhibit data redundancy. Indeed, each of the three $l\_name$ occurrences of ‘Futrell’ in the first three tuples in Table 7.1 is redundant. However, such redundant occurrences are intrinsically linked to the requirement that the tuples must be complete on $date\_register$, since the eFD does not apply otherwise. This link between data redundancy and data completeness requirements encoded explicitly in the eFDs. In what follows, we will develop a full-fledged normalization framework that that tailors classical schema design to data completeness requirements.

### 7.3.2 Axiomatic Characterization

This section establishes axiomatic and algorithmic characterizations of the implication problem for eUCs and eFDs. The linear-time dependability we establish is important for the schema design framework we will develop subsequently.
Let $\Sigma \cup \{ \varphi \}$ denote a set of eUCs and eFDs over relation schema $R$. We say that $\Sigma$ implies $\varphi$, iff every relation over $R$ that satisfies all $\sigma \in \Sigma$ also satisfies $\varphi$. The implication problem for a class $C$ of constraints is to decide, for arbitrary $R$ and $\Sigma \cup \{ \varphi \}$ in $C$, whether $\Sigma$ implies $\varphi$.

Firstly, we would like to obtain an axiomatization for eUCs and eFDs which extends the well known Armstrong axioms [9]. An axiomatization does not only help us understand the interaction of the constraints, but also forms the basis for our syntactic normal form proposals later. The definitions of inference from a system $E (\models E)$, as well as sound and complete sets of inference rules are standard (see Chapter 2).

Table 7.2 shows three axiomatizations. The top box is for eUCs alone, which has been proved in Chapter 5. The middle box is for eFDs alone and the last box is the interaction rule between eUCs and eFDs. All boxes together form the axiomatization $E$ for eUCs and eFDs. The concept of closure is often helpful for proving axiomatization.

**Definition 7.2 (E-closure)** Let $\Sigma$ be a set of eUCs and eFDs over relation schema $R$ and $X \subseteq E \subseteq R$. The closure of $X$ with respect to the data completeness requirement $E$ and $\Sigma$ is the set $X_{E, \Sigma}^+ = \{ A \in E \mid \Sigma \models E : X \rightarrow A \}$.
Lemma 7.3.1 (Soundness of $E$). The rules in $E$ are sound for the implication of eUCs and eFDs.

Proof. We show soundness for each rule of $E$ in turn.

The (trivial key)-rule is sound since the scope $r^R$ of every relation $r$ over schema $R$ with respect to $R$ is a relation, and can therefore not contain two different tuples with matching values on all the attributes in $R$.

For the soundness of the (eUC extension)-rule, we assume that a relation $r$ over relation schema $R$ violates the eUC $EE' : UU'$. That is, there are two different $EE'$-total tuples $t, t' \in r^{EE'}$ such that $t(UU') = t'(UU')$ holds. Since $r^{EE'} \subseteq r^E$ and $U \subseteq UU'$ hold, it follows that there are two different $E$-total tuples $t, t' \in r^E$ such that $t(U) = t(U')$ holds, Consequently, the relation $r$ also violates the eUC $E : U$.

The (trivial eFD)-rule is sound since the scope $r^E$ of every relation $r$ over relation schema $R$ with respect to $E$ satisfies the trivial FD $XY \rightarrow Y$. Note that the latter statement follows from the soundness of the reflexivity rule for traditional functional dependencies.

The (eFD extension)-rule is sound since the scope $r^E$ of every relation $r$ over relation schema $R$ with respect to $E$ satisfies the FD $X \rightarrow XY$ whenever it satisfies the FD $X \rightarrow Y$. The latter statement follows from the soundness of the extension rule for traditional functional dependencies.

For the soundness of the (eFD transitivity)-rule, we assume that a relation $r$ over relation schema $R$ violates the eFD $EE' : X \rightarrow Z$. That is, there are two different $EE'$-total tuples $t, t' \in r^{EE'}$ such that $t(X) = t'(X)$ and $t(Z) \neq t'(Z')$ hold. Due to the soundness of the transitivity rule for traditional functional dependencies, it follows that $r^{EE'}$ violates the FD $X \rightarrow Y$ or the FD $Y \rightarrow Z$. Since $r^{EE'} \subseteq r^E$ and $r^{EE'} \subseteq r^{EE'}$ hold, it follows that $r^E$ violates the FD $X \rightarrow Y$ or $r^E$ violates the FD $Y \rightarrow Z$. Consequently, $r$ violates the eFD $E : X \rightarrow Y$ or the eFD $E' : Y \rightarrow Z$.

For the soundness of the (eUC to eFD)-rule, we assume that a relation $r$ over relation schema $R$ violates the eFD $E : X \rightarrow E$. That is, there are two
E-total tuples $t, t' \in r^E$ such that $t(X) = t'(X)$ and $t(E) \neq t'(E)$ hold. In particular, $t$ and $t'$ are different tuples. Consequently, there are two different E-total tuples $t, t' \in r^E$ such that $t(X) = t'(X)$ holds. That is, the relation $r$ over relation schema $R$ violates the eUC $E : X$.

For the soundness of the (eUC pullback)-rule, we assume that a relation $r$ over relation schema $R$ violates the eUC $E : X$. That is, there are two different E-total tuples $t, t' \in r^E$ such that $t(X) = t'(X)$ holds. If $r$ violates the eFD $E : X \rightarrow Y$, then we are done. Otherwise, $r^E$ satisfies the FD $X \rightarrow Y$. In particular, $t(Y) = t'(Y)$ and therefore we have two different E-total tuples $t, t' \in r^E$ such that $t(XY) = t'(XY)$ holds. Hence, the eUC $E : XY$ is violated.

**Lemma 7.3.2 (Extra Rules).** Inference rules from Table 7.3 are sound for the implication of eUCs and eFDs.

**Proof.** The (eFD decomposition)-rule can be inferred from the rules in $\mathcal{E}$ as follows:

\[
\begin{align*}
E : X \rightarrow YZ & \quad E : YZ \rightarrow Y \\
\hline \\
E : X \rightarrow Y & 
\end{align*}
\]

The (eFD union)-rule can be inferred from the rules in $\mathcal{E}$ as follows:

\[
\begin{align*}
E : XY \rightarrow X & \quad E : X \rightarrow Z \\
E : X \rightarrow Y & \\
\hline \\
E : X \rightarrow XY & \quad E : XY \rightarrow XYZ & \quad E : XYZ \rightarrow YZ \\
\hline \\
E : X \rightarrow YZ & 
\end{align*}
\]

The (eFD addendum)-rule can be inferred from the rules in $\mathcal{E}$ as follows:

\[
\begin{align*}
E : X \rightarrow Y & \quad EE' : Y \rightarrow Y \\
\hline \\
EE' : X \rightarrow Y & 
\end{align*}
\]

**Theorem 7.3.3 (Soundness and Completeness of $\mathcal{E}$).** $\mathcal{E}$ is a sound and complete axiomatization for the implication of eUCs and eFDs.
Proof. The soundness of the rules in $E$ has been established in Lemma 7.3.1. It remains to show completeness. For this purpose we proceed classically by contraposition, and assume for arbitrarily given relation schema $R$, and an eUC and eFD set $\Sigma \cup \{ \phi \}$ over $R$ such that $\phi \not\in \Sigma_+^E$ holds. We need to show that $\phi \not\in \Sigma_+^*$ holds. For that purpose, we will construct a relation $r$ over $R$ such that $r$ satisfies all eUCs and eFDs in $\Sigma$, but does not satisfy $\phi$.

Let $\phi$ denote either the eUC $E : X$ with respect to $E$ and $\Sigma$. The soundness of the (eFD union)-rule, established in Lemma 7.3.2, shows that $E : X \to X_{E,\Sigma}^+ \in \Sigma_+^E$ holds.

Let $r = \{ t, t' \}$ be a relation over $R$ such that $t(A) = 0$ for all $A \in R$ and $t'(A) = \begin{cases} 0 & \text{if } A \in X_{E,\Sigma}^+ \\ 1 & \text{if } A \in E - X_{E,\Sigma}^+ \\ \bot & \text{if } A \in R - E \end{cases}.$

The relation $r$ is as follows.

<table>
<thead>
<tr>
<th>$X_{E,\Sigma}^+$</th>
<th>$E - X_{E,\Sigma}^+$</th>
<th>$R - E$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0...0</td>
<td>0...0</td>
<td>0...0</td>
</tr>
<tr>
<td>0...0</td>
<td>1...1</td>
<td>$\bot$...$\bot$</td>
</tr>
</tbody>
</table>

**Case 1.** Here, $\phi = E : X$. If $E = R$ and $X_{E,\Sigma}^+ = R$, then $R : R$ and $R : X \to X_{E,\Sigma}^+ = R : X \to R \in \Sigma_+^E$, and thus also $R : X = E : X \in \Sigma_+^E$. This would be a contradiction to our assumption that $E : X \not\in \Sigma_+^E$. Hence, $E \subset R$ or $X_{E,\Sigma}^+ \subset R$. That is $R - E \neq \emptyset$ or $(E - X_{E,\Sigma}^+ \neq \emptyset$ if $E = R$). That is, $t = t'$ and $r$ is guaranteed to be a two-tuple relation by construction.

Since $X \subseteq X_{E,\Sigma}^+$ holds and $r^E = r$, it follows from the construction of $r$ that $r$ violates $E : X$. It therefore remains to show in this case that $r$ satisfies every eUC and every eFD in $\Sigma$. Let $\sigma$ denote such an element of $\Sigma$.

**Case 1.a** Here, $\sigma$ denotes the eUC $E' : X' \in \Sigma$. Assume, to the contrary, that $r$ violates $\sigma$. It follows from the construction of $r$ that $E' \subseteq E$ and $X' \subseteq X_{E,\Sigma}^+$ both hold. Applying the (eUC extension)-rule to $E' : X' \in \Sigma$ gives us $E : X_{E,\Sigma}^+ \in \Sigma_+^E$. Since $E : X \to X_{E,\Sigma}^+ \in \Sigma_+^E$ holds, we can apply the (eUC pullback)-rule to infer $E : X \in \Sigma_+^E$. This is a contradiction to our assumption
that \( \varphi = E : X \not\in \Sigma^+_\ell \). Consequently, our assumption that \( r \) violates \( \sigma \) must have been wrong, and we conclude that \( r \) satisfies \( \sigma \) in this case.

**Case 1.b** Here, \( \sigma \) denotes the eFD \( E' : X' \to Y' \in \Sigma \). Assume again, to the contrary, that \( r \) violates \( \sigma \). It follows from the construction \( r \) that \( E' \subseteq E, X' \subseteq X^+_{E,\Sigma} \), and \( Y' \cap (E - X^+_{E,\Sigma}) \neq \emptyset \) all hold. From \( E : X \to X^+_{E,\Sigma} \in \Sigma^+_\ell \) and \( E' \subseteq X^+_{E,\Sigma} \), we infer \( E : X \to X' \in \Sigma^+_\ell \) by applying the (eFD transitivity)-rule from Lemma 7.3.2. Applying the (eFD transitivity)-rule to \( E : X \to X' \in \Sigma^+_\ell \) and \( E' : X' \to Y' \in \Sigma \), we infer \( EE' : X \to Y' \in \Sigma^+_\ell \). Since \( E' \subseteq E \), we actually have \( E : X \to Y' \in \Sigma^+_\ell \). According to the definition of the attribute set closure, we must then have that \( Y' \subseteq X^+_{E,\Sigma} \). This, however, is a contradiction to \( Y' \cap (E - X^+_{E,\Sigma}) \neq \emptyset \). Consequently, our assumption that \( r \) violates \( \sigma \) must have been wrong, and we conclude that \( r \) satisfies \( \sigma \) in this case.

**Case 2.** Here, \( \varphi = E : X \to Y \). If \( X^+_{E,\Sigma} = R \), then \( E = R \) and \( E : X \to X^+_{E,\Sigma} \in \Sigma^+_\ell \) would mean that \( R : X \to R \in \Sigma^+_\ell \). The (trivial eFD)-rule means that \( R : R \to Y \in \Sigma^+_\ell \), and applying the (eFD transitivity)-rule to \( R : X \to R \in \Sigma^+_\ell \) and \( R : R \to Y \in \Sigma^+_\ell \), we infer \( R : X \to Y \in \Sigma^+_\ell \). Since \( E = R \), we would then derive the contradiction that \( \varphi = E : X \to Y \in \Sigma^+_\ell \). Consequently, \( X^+_{E,\Sigma} \subseteq R \). Hence, \( E - X^+_{E,\Sigma} \neq \emptyset \) or \( (R - E \neq \emptyset \) if \( E = X^+_{E,\Sigma} \). Thus, \( t \neq t' \) and \( r \) is a two-tuple relation in this case, too.

If \( Y \subseteq X^+_{E,\Sigma} \), then \( E : X \to Y \in \Sigma^+_\ell \) contrary to our assumption. Hence, \( Y \cap (E - X^+_{E,\Sigma}) \neq \emptyset \). Since \( X \subseteq X^+_{E,\Sigma} \), \( Y \cap (E - X^+_{E,\Sigma}) \neq \emptyset \) and \( r^E = r \), it follows from the construction of \( r \) that \( r \) violates \( E : X \to Y \). It therefore remains to show in this case that \( r \) satisfies every eUC and every eFD in \( \Sigma \).

**Case 2.a** Here, \( \sigma \) denotes the eUC \( E' : X' \in \Sigma \). Assume, to the contrary, that \( r \) violates \( \sigma \). It follows from the construction of \( r \) that \( E' \subseteq E \) and \( X' \subseteq X^+_{E,\Sigma} \) both hold. Applying the (eUC extension)-rule to \( E' : X' \in \Sigma \) gives us \( E : X^+_{E,\Sigma} \in \Sigma^+_\ell \). Since \( E : X \to X^+_{E,\Sigma} \in \Sigma^+_\ell \) holds, we can apply the (eUC pullback)-rule to infer \( E : X \in \Sigma^+_\ell \). From the (eUC to eFD)-rule we can then infer \( E : X \to E \in \Sigma^+_\ell \), and since \( E : E \to Y \in \Sigma^+_\ell \), we can apply the (eFD
7.3. Foundations

<table>
<thead>
<tr>
<th>$E : X \rightarrow YZ$</th>
<th>$E : X \rightarrow Y$ $E : X \rightarrow Z$</th>
<th>$E : X \rightarrow Y$ $EE' : X \rightarrow Y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>(eFD decomposition)</td>
<td>(eFD union)</td>
<td>(eFD addendum)</td>
</tr>
</tbody>
</table>

**Table 7.3: Inference rules from $E$**

*transitivity*)-rule to $E : X \rightarrow E \in \Sigma^+_E$ and $E : E \rightarrow Y \in \Sigma^+_E$ to infer the contradiction that $E : X \rightarrow Y \in \Sigma^+_E$. Hence, our assumption must have been wrong, and $r$ satisfies $\sigma$ in this case.

**Case 2.b** Here, $\sigma$ denotes the eFD $E' : X' \rightarrow Y' \in \Sigma$. Assume again, to the contrary, that $r$ violates $\sigma$. It follows from the construction of $r$ that $E' \subseteq E$, $X' \subseteq X^+_E$, and $Y' \cap (E - X^+_E) \neq \emptyset$ all hold. From $E : X \rightarrow X^+_E \in \Sigma^+_E$ and $X' \subseteq X^+_E$, we infer $E : X \rightarrow X' \in \Sigma^+_E$ by applying the (eFD decomposition)-rule from Lemma 7.3.2. Applying the (eFD transitivity)-rule to $E : X \rightarrow X' \in \Sigma^+_E$ and $E' : X' \rightarrow Y' \in \Sigma$, we infer $EE' : X \rightarrow Y' \in \Sigma^+_E$. Since $E' \subseteq E$, we actually have $E : X \rightarrow Y' \in \Sigma^+_E$. According to the definition of the attribute set closure, we must then have that $Y' \subseteq X^+_E$. This, however, is a contradiction to $Y' \cap (E - X^+_E) \neq \emptyset$. Consequently, our assumption that $r$ violates $\sigma$ must have been wrong, and we conclude that $r$ satisfies $\sigma$ in this case.

This concludes the completeness proof.

Table 7.3 shows the rules that follow $E$.

**Example 7.1 (Implication of EUCs and EFDs)** Consider our running example where $R = \{f, l, p, d\}$ and $\Sigma = \{fld : fld, dpl : p \rightarrow l\}$. Then $\Sigma$ implies the eUC $\varphi = dflp : dfp$ as the following inference shows:

$$
\frac{dflp : dfp \rightarrow p}{dflp : dfp \rightarrow l}
\frac{dflp : dfp \rightarrow l}{dflp : dfp \rightarrow dfp}
$$
However, the eFD \( f : p \rightarrow l \) is not implied by \( \Sigma \) because of the last two tuples in Table 7.1.

### 7.3.3 Algorithmic Characterization

Reasoning efficiently about eUCs and eFDs will help us decide if a given schema meets a normal form condition, or transform the schema into one that meets the condition. In the relational model, FD implication can be decided in linear time. We will achieve the same for eUCs and eFDs.

The technical underpinning of our framework translates every set of eUCs and eFDs into a set of FDs such that existing algorithms for deciding FD implication can be used to decide implication of eUCs and eFDs. The translation is given next.

**Definition 7.3 ((E, FD)-Reduct)**  For a given set \( \Sigma \) of eUCs and eFDs over relation schema \( R \), and a given attribute set \( E \subseteq R \), the \((E, \text{FD})\)-reduct is \( \Sigma[E] = \{ X \rightarrow R \mid \exists E' \subseteq E(E' : X \in \Sigma) \} \cup \{ X \rightarrow Y \mid \exists E' \subseteq E(E' : X \rightarrow Y \in \Sigma) \} \).

The importance of the \((E, \text{FD})\)-reduct is embodied in an algorithmic characterization of our implication problem.

**Theorem 7.3.4 ((E, FD)-Reduct).** Let \( \Sigma \cup \{ E : X, E : X \rightarrow Y \} \) denote a set of eUCs and eFDs over relation schema \( R \). Then:

1. \( \Sigma \vdash E : X \rightarrow Y \) if and only if \( \Sigma[E] \vdash X \rightarrow Y \)

2. \( \Sigma \models E : X \) if and only if \( E = R = X \) or there is some \( E' : X' \in \Sigma \) such that \( E' \subseteq E \) and \( X' \subseteq X_{\Sigma[E']}^* \).

**Proof.** We start by proving (1), and show first that \( \Sigma \models E : X \rightarrow Y \) implies \( \Sigma[E] \models X \rightarrow Y \). For the purpose we proceed by contraposition, assuming that \( \Sigma[E] \not\models X \rightarrow Y \). Consequently there is some total two-tuple relation \( r = \{ t, t' \} \) over \( R \) such that \( r \) violates \( X \rightarrow Y \) and \( r \) satisfies every FD in \( \Sigma[E] \).
We now define the $E$-complete two-tuple relation $r_E = \{ t, t'_E \}$ over $R$ where, for all $A \in R$,
\[
t'_E(A) = \begin{cases} 
    t'(A), & \text{if } A \in E \\
    \bot, & \text{otherwise}
\end{cases}
\]

Since $r$ violates $X \rightarrow Y$, it follows that $r_E^E = r_E$ violates $X \rightarrow Y$ since $X, Y \subseteq E$ and the tuple $t$ is unchanged and $t'_E(A) = t'(A)$ for all attributes $A \in E$. It remains to show that $r_E^E = r_E$ satisfies every $\sigma \in \Sigma$. For all $E' \subseteq R$ where $E' - E \neq \emptyset$ if follows that $r_E^{E'} = \{ t \}$, and it therefore follows that $r_E$ satisfies every $\sigma \in \Sigma$ where $\sigma = E' : X' \rightarrow Y'$ with $E' \subseteq E$, then $X' \rightarrow Y' \in \Sigma[E]$. In this case, however, $r_E$ satisfies $\sigma$ since $r$ satisfies $X' \rightarrow Y'$. This leaves us to consider the case where $\sigma = E' : X'$ with $X' \subseteq E' \subseteq E$. If $r_E$ violates $\sigma$, then $t(X') = t'_E(X')$. However, $X' \rightarrow R \in \Sigma[E]$ holds in this case, and since $r$ satisfies $X' \rightarrow R$, we could arrive at the contradiction that $t = t'$. Consequently, $r_E$ also satisfies $\sigma$ in this case. We have therefore shown that $\Sigma \not| \neq E : X \rightarrow Y$ holds.

We will now show that $\Sigma[E] \models X \rightarrow Y$ implies $\Sigma \models E : X \rightarrow Y$. For that purpose we proceed by contraposition, assuming that $\Sigma \not| \neq E : X \rightarrow Y$. Hence, there is an $E$-complete two-tuple relation $r_E = \{ t_E, t'_E \}$ such that $r_E$ violates $E : X \rightarrow Y$ and $r_E$ satisfies every $\sigma \in \Sigma$. Define $r = \{ t, t' \}$ where
\[
t(A) = \begin{cases} 
    t_E(A), & \text{if } A \in E \\
    0, & \text{otherwise}
\end{cases}
\]
and
\[
t'(A) = \begin{cases} 
    t'_E(A), & \text{if } A \in E \\
    1, & \text{otherwise}
\end{cases}
\]

It follows that $r$ is a total two-tuple relation over $R$, that $r$ violates $X \rightarrow Y$ since $X, Y \subseteq E$ and $r_E^E$ violates $X \rightarrow Y$, and that $r$ satisfies every $X' \rightarrow R$, $X' \rightarrow Y' \in \Sigma[E]$ since $r_E$ satisfies every $E' : X', E' : X' \rightarrow Y' \in \Sigma$ where $E' \subseteq E$. It follows that $\Sigma[E] \not| \neq X \rightarrow Y$, which draws a contradiction. This
completes the proof of (1).

We will now prove (2), and show first directly that \( \Sigma \models E : X \) is implied by a) \( E = R = X \), or b) there is some \( E' : X' \in \Sigma \) such that \( E' \subseteq E \) and \( X' \subseteq X^+_\Sigma[E] \). If a) \( E = R = X \) holds, then the soundness of the (trivial key)-rule shows that \( \Sigma \models E : X \). If b) holds, then (1) shows that \( E : X \to X' \) is implied by \( \Sigma \) since \( E' \subseteq E \) and \( X' \subseteq X^+_\Sigma[E] \). From \( E' : X' \in \Sigma \) we conclude \( \Sigma \models E : XX' \) by soundness of the (eUC extension)-rule. Form \( \Sigma \models E : XX' \) and \( \Sigma \models E : X \to X' \) we conclude \( \Sigma \models E : X \) by soundness of the (eUC pullback)-rule.

It remains to show that \( \Sigma \models E : X \) implies a) or b). For that purpose we proceed by contraposition, assuming that neither a) nor b) hold. Since \( E \neq R \neq X \) we conclude that \( E \) is a proper subset of \( R \). Since b) does not hold, we conclude that for every \( E' : X' \in \Sigma \) where \( E' \subseteq E \) holds, \( X' \) is not a subset of \( X^+_\Sigma[E] \). Let \( r = \{ t, t' \} \) denote the following \( E \)-complete two-tuple relation over \( R \):

<table>
<thead>
<tr>
<th>( X^+_\Sigma[E] )</th>
<th>( E - X^+_\Sigma[E] )</th>
<th>R-E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0...0</td>
<td>0...0</td>
<td>0...0</td>
</tr>
<tr>
<td>0...0</td>
<td>1...1</td>
<td>⊥...⊥</td>
</tr>
</tbody>
</table>

Since \( X \subseteq X^+_\Sigma[E] \) it follows immediately that \( r \) violates the eUC \( E : X \). It remains to show that \( r \) satisfies every \( \sigma \in \Sigma \). Let us look first at the case where \( \sigma = E' : X' \). If \( E' \not\subseteq E \), then \( r^{E'} \) consists of only one tuple, so it satisfies \( \sigma \). If \( E' \subseteq E \), then \( X' \) intersects non-trivially with \( E - X^+_\Sigma[E] \) since b) does not hold. Consequently, \( t(X') \neq t'(X') \). Hence, \( r \) satisfies \( \sigma \) in this case, too. Let us now look at the case where \( \sigma = E' : X' \to Y' \). If \( E' \not\subseteq E \), then \( r^{E'} \) consists of only one tuple, so it satisfies \( \sigma \). It remains to consider the case where \( E' \subseteq E \). We show the following: if \( t(X') = t'(X') \), then \( t(Y') = t'(Y') \). Indeed, if \( t(X') = t'(X') \), then \( X' \subseteq X^+_\Sigma[E] \) and by (1) we conclude \( \Sigma \models E : X \to X' \). From \( E' : X' \to Y' \in \Sigma \) and \( \Sigma \models E : X \to X' \) we conclude that \( \Sigma \models EE' : X \to Y' \) by the (eFD transitivity)-rule. Since \( E' \subseteq E \) holds, we conclude that \( \Sigma \models E : X \to Y' \). According to (1) we
obtain that $Y' \subseteq X^+_{\Sigma[E]}$ holds. Consequently, the construction of $r$ shows that $t(Y') = t'(Y')$ holds, too. We conclude that $r$ satisfies every $\sigma \in \Sigma$. This completes the proof of (2) \hfill \square

Theorem 7.3.4 suggests Algorithm 19 for deciding implication. If $\varphi = R : R$, the implication must be true (line 3-4). Otherwise, standard algorithms compute the closure of the attribute set given $\Sigma[E]$ (line 5). If $\varphi$ is an eUC and (2) in Theorem 7.3.4 is met, then the implication is true (line 6-7). If $\varphi$ is an eFD and (1) in Theorem 7.3.4 is met, then the implication is true (line 8-9). Otherwise, the implication is false.

Algorithm 19 Decide Implication

1: Input: A set $\Sigma \cup \{ \varphi \}$ of eUCs and eFDs over relation schema $R$
2: Output: $\begin{cases} 
\text{true}, & \text{if } \Sigma \models E'[d] \\
\text{false}, & \text{if otherwise}
\end{cases}$
3: if $\varphi = R : R$ then
4: return $\text{true}$
5: Compute $\Sigma^+_{\Sigma[E]}$ \hfill \triangleright FD attribute set closure algorithm
6: if $\varphi = E : X \land \exists E' : X' \in \Sigma (E' \subseteq E \land X' \subseteq X^+_{\Sigma[E]})$ then
7: return $\text{true}$
8: if $\varphi = E : X \rightarrow Y \land Y \subseteq Y \subseteq X^+_{\Sigma[E]}$ then
9: return $\text{true}$
10: return $\text{false}$

The soundness of Algorithm 19 follows from Theorem 7.3.4, linear time decidability from that of FD implication [15], and PTIME-hardness from a reduction of HORN-SAT [36], [55].

Corollary 7.3.5 (Complexity of Embedded Implication). The implication problem of eUCs and eFDs is PTIME-complete. On input $(\Sigma \cup \{ \varphi \}, R)$, Algorithm 19 decides the implication problem $\Sigma \models \varphi$ in time $O(|\Sigma \cup \{ \varphi \}|)$.

Example 7.2 (Implication of EUCs and EFDs) In running example $R = \{f, l, p, d\}$ and $\Sigma = \{fld :fld, dpl : p \rightarrow l\}$, $\Sigma$ implies $\varphi = dflp : dfp$ since $\Sigma[dflp] = \{p \rightarrow l, dfl \rightarrow p\}$, there is some eUC $E' = dfl : dfl = X' \in \Sigma$ such that $E' \subseteq E$, and $X' \subseteq (dfp)^+_{\Sigma[dflp]} = dflp$, which means (2) of Theorem 7.3.4 is met. The eFD $f : p \rightarrow l$ is not implied by $\Sigma$ as $\Sigma[E] = \Sigma[flp] = \varnothing$, and $p \rightarrow l$ is
not implied by $\Sigma$ as $\Sigma[E] = \Sigma[fp] = \emptyset$, and $p \rightarrow l$ is not implied by $\Sigma[E]$, which means (1) of Theorem 7.3.4 is not met.

7.4 Normal Forms

Our goal is to tailor classical schema design to data completeness requirements of applications. For that purpose, we stipulate the semantic normal form condition that no redundant data values can occur in any $E$-complete ($E$-total) records on any relations that satisfy a given set of eUCs and eFDs. We will characterize this condition by generalizing the well-known BCNF. Similarly, we further generalize the well-known 3NF to characterize the minimization of data redundancy in $E$-complete records across all dependency-preserving decompositions.

7.4.1 $E$-Redundancy Free Normal Form

Motivated by our introductory examples, we propose notions of data redundancy that are tailored towards the requirements of records in terms of their completeness. For this, we generalize the classical proposal (see Definition 2.22 and Definition 2.23 in Chapter 2) by Vincent [104] as follows. Firstly, recall the definition of replacement of a data value.

**Definition 7.4 (Replacement)** For a relation schema $R$, attribute $A$ of $R$, tuple $t$ over $R$, and set $\Sigma$ of constraints over $R$, a replacement of $t(A)$ is a tuple $\bar{t}$ over $R$ such that for all $\bar{A} \in R - \{A\}$ we have $\bar{t}(\bar{A}) = t(\bar{A})$, and $\bar{t}(A) \neq t(A)$.

**Definition 7.5 ($E$-RFNF)** Let $R$ denote a relation schema, $E \subseteq R$, $\Sigma$ a set of constraints over $R$, $A \in E$ an attribute, $r$ a relation over $R$ that satisfies $\Sigma$, and $t$ a tuple in $r^E$. An $E$-replacement of $t(A)$ is a replacement of $t(A)$ that is $E$-complete. The data value occurrence $t(A)$ is $E$-redundant for $\Sigma$ if and only if for every $E$-replacement $\bar{t}$ of $t(A)$, $\bar{r} = (r - \{t\}) \cup \{\bar{t}\}$ violates some constraint in $\Sigma$. $R$ is
in E-Redundancy-Free Normal Form (E-RFNF) for \( \Sigma \) if and only if there are no relation \( r \) over \( R \) that satisfies \( \Sigma \), tuple \( t \in r^E \), and attribute \( A \in E \), such that the data value occurrence \( t(A) \) is E-redundant for \( \Sigma \).

\[ \]

**Example 7.3 (E-redundant)** In our running example \( R = \text{flpd} \) and \( \Sigma = \{ \text{fld} : \text{fld}, p : p \rightarrow l \} \), the relation in Table 7.1 shows that \( R \) is not in dlp-RFNF for \( \Sigma \): every dlp-replacement for either of the bold occurrences would violate the eFD \( d : p \rightarrow l \).

While E-RFNF is independent of the type of constraints, we will assume from now on that \( \Sigma \) is a set of eUCs and eFDs. As our first result we characterize the E-RFNF for \( \Sigma \) in terms of the RFNF for the \((E, FD)\)-reduct \( \Sigma[E] \).

**Theorem 7.4.1 (E-RFNF).** For all sets \( \Sigma \) of eUCs and eFDs over \( R \), \( R \) is in E-RFNF for \( \Sigma \) if and only if \( R \) is in RFNF for \( \Sigma[E] \).

**Proof.** We show first the following: if \( R \) is not in E-RFNF for \( \Sigma \), then \( R \) is not in RFNF for \( \Sigma[E] \). According to our hypothesis, there is some relation \( r \) over \( R \) that satisfies \( \Sigma \), an attribute \( A \in E \), and a tuple \( t \in r^E \) such that \( t(A) \) is E-redundant for \( \Sigma \). Hence, for every E-replacement \( \bar{t} \) of \( t(A) \), \( \bar{r} = (r - \{ t \}) \cup \{ \bar{t} \} \) violates some eUCs or eFD in \( \Sigma \). Consequently, there must be some tuple \( t' \in r^E \) such that \( t \neq t' \), \( t(X'A) = t'(X'A) \) and \( A \in Y' - X' \) for some non-trivial eFD \( E' : X' \rightarrow Y' \in \Sigma \) with \( E' \subseteq E \). Let \( r_C = \{ t_C, t'_C \} \) be the complete relation where \( t_C(A') = t(A') \) whenever \( t(A') \neq \bot \), \( t'_C(A') = t'(A') \) whenever \( t'(A') \neq \bot \), otherwise \( t_C(A'), t'_C(A') \) can be arbitrary complete values as long as \( t_C(A') \neq t'_C(A') \). In particular, \( t_C(E) = t(E) \) and \( t'_C(E) = t'(E) \) since \( t, t' \in r^E \). Since \( t \neq t' \) it follows that \( t_C \neq t'_C \). Consequently, \( r_C \) is a two-tuple \( R \)-relation that satisfies \( \Sigma[E] \) since \( \{ t, t' \} \) is a two-tuple \( E \)-complete relation that satisfies \( \Sigma \). The data value occurrence \( t_C(A) \) in \( r_C \) is redundant for \( \Sigma[E] \) since every replacement \( \bar{t} \) of \( t(A) \), \( \bar{r} = (r - \{ t \}) \cup \{ \bar{t} \} \) violates the FD \( X' \rightarrow Y' \in \Sigma[E] \). Hence, \( R \) is not in RFNF for \( \Sigma[E] \).
We now show: if $R$ is not in RFNF for $\Sigma$, then $R$ is not in $E$-RFNF for $\Sigma$. According to our hypothesis, there is some complete relation $r$ over $R$ that satisfies $\Sigma$, some tuple $t \in r$, and attribute $A \in R$ such that the data value occurrence $t(A)$ is redundant for $\Sigma$. That is, for every replacement $\bar{t}$ of $t(A)$, $\bar{r} = (r - \{t\}) \cup \{\bar{t}\}$ violates some constraint in $\Sigma$. Consequently, there must be some tuple $t' \in r$ such that $t \neq t'$, $t(X' \backslash A) = t'(X' \backslash A)$ and $A \in Y' - X'$ for some FD $X' \rightarrow Y' \in \Sigma$. In particular, $Y' \subseteq R$ as otherwise $t = t'$. Hence, there is some $E': X' \rightarrow Y' \in \Sigma$. In particular, $Y' \subseteq E' \subseteq E$. Let $r_1 = \{t, t'_1\}$ such that $t'_1(A') = t'(A')$ whenever $A' \in E$, and $t'(A') = \perp$ whenever $A' \not\in E$. Since $r$ satisfies $\Sigma$, $r_1$ satisfies $\Sigma$. In summary, there is an $E$-complete relation $r_1$ over $R$ that satisfies $\Sigma$, an attribute $A \in E$, and a tuple $t \in r_1$ such that every $E$-replacement $\bar{t}$ of $t(A)$, $\bar{r}_1 = (r_1 - \{t\}) \cup \{\bar{t}\}$ violates $E' : X' \rightarrow Y' \in \Sigma$. Hence, $R$ is not in $E$-RFNF for $\Sigma$.

Example 7.4 ($E$-RFNF) In our running example $R = flpd$, $\Sigma = \{fld : fld, dpl : p \rightarrow l\}$ and $E = dlp$, we would get $\Sigma[E] = \{p \rightarrow l\}$. That is, $R$ is also not in RFNF for $\Sigma[E]$.

7.4.2 $E$-BCNF

We now characterize the semantic $E$-RFNF by purely syntactic means. For that purpose, we generalize the BCNF condition (see Definition 2.24) to accommodate completeness requirements.

Definition 7.6 ($E$-BCNF) For relation schema $R$ and $E \subseteq R$, $R$ is in $E$-BCNF for a set $\Sigma$ of eUCs and eFDs over $R$ if and only if for every eFD $E : \rightarrow Y \in \Sigma^+_E$ where $Y \not\subseteq X$ and $E : X \in \Sigma^+_E$.

Example 7.5 ($E$-BCNF) In our running example $R = flpd$ and $\Sigma = \{fld : fld, dpl : p \rightarrow l\}$, $R$ is not in dlp-BCNF for $\Sigma$, since the eFD $dpl : p \rightarrow l \in \Sigma$, $\{l\} \not\subseteq \{p\}$, but $dlp : p \not\in \Sigma^+_E$. 

\[ \]

\[ \]

\[ \]

\[ \]
Recall that sets $\Sigma$ and $\Theta$ are *covers* of one another if $\Sigma^* = \Theta^*$ holds. The property of being in $E$-BCNF for $\Sigma$ is independent of the representation of $\Sigma$. That is, for any cover $\Theta$ of $\Sigma$, $R$ is in $E$-BCNF for $\Sigma$ iff $R$ is in $E$-BCNF for $\Theta$. The $E$-BCNF condition for an eUC/eFD set $\Sigma$ can be characterized by the BCNF condition for the FD set $\Sigma|E$.

**Lemma 7.4.2 (EUC Reduction).** Let $\Sigma$ be a set of eUCs and eFDs over relation schema $R$, and $E \subseteq R$ an attribute subset of $R$. Then $\Sigma|E \models X \rightarrow R$ if and only if $\Sigma \models R : X \rightarrow R$. This shows the lemma for the case where $E = R$.

Assume now that $E$ is a proper subset of $R$. We show that $\Sigma|E \models X \rightarrow R$ if and only if ($\ast$) there is some $E' : X' \in \Sigma$ such that $E' \subseteq E$ and $\Sigma \models E : X \rightarrow X'$. The second property of Theorem 7.3.4 then ensures that ($\ast$) is equivalent to $\Sigma \models E : X$.

If ($\ast$) holds, then $X' \rightarrow R \in \Sigma|E$ by definition of $\Sigma|E$ and $\Sigma|E \models X \rightarrow X'$ by the first property of Theorem 7.3.4. Consequently, $\Sigma|E \models X \rightarrow R$ by the soundness of the transitivity rule for traditional FDs. Vice versa, assume that ($\ast$) does not hold. That is, for all $X' \rightarrow R \in \Sigma|E$ we have $X' \not\subseteq X^+_\Sigma|E$. We will show that $X \rightarrow R$ is not implied by $\Sigma|E$. Let $r = \{t, t'\}$ denote the following complete two-tuple relation over $R$.

| $X^+_{\Sigma|E}$ | $R - X^+_{\Sigma|E}$ |
|-----------------|---------------------|
| 0...0           | 0...0               |
| 0...0           | 1...1               |

For $X' \rightarrow R$ to be in $\Sigma|E$ there are two possibilities: 1) there is some $E' : X' \rightarrow R \in \Sigma$ such that $E' \subseteq E$, and 2) there is some $E' : X' \in \Sigma$ such that $E' \subseteq E$. For 1) it follows that $R \subseteq E' \subseteq E$, that is $R = E' = E$ which would contradict our assumption that $E \subset R$. Hence, only 2) is possible by assumption. Furthermore, for every $X' \rightarrow Y \in \Sigma|E$ with $Y \neq R$ it follows
that there is some $E' : X' \rightarrow Y \in \Sigma$ such that $E' \subseteq E$. Hence, $X'Y \subseteq E$. That is, 3) for every $X' \rightarrow Y \in \Sigma[E]$ with $Y \neq R$ it follows that $X'Y \subseteq E$.

Since $X \subseteq E$ and $E$ is assumed to be a proper subset of $R$ it follows that $X$ is a proper subset of $R$. Due to (3) we conclude that $X^+_\Sigma[E] \subseteq E$ is also a proper subset of $R$. Hence, $r$ is a two-tuple relation, namely $R - X^+_\Sigma[E] \neq \emptyset$. Since $X \subseteq X^+_\Sigma[E]$ we conclude that $X \rightarrow R$ is not satisfied by $r$. We now show that $r$ satisfies every FD $U \rightarrow V \in \Sigma[E]$. Assume that $t(U) = t'(U)$ as otherwise there is nothing to show. It follows that $U \subseteq X^+_\Sigma[E]$ and therefore $X \rightarrow U \in \Sigma[E]^+_A$ where $A$ is the Armstrong axiom [9]. From $U \rightarrow V \in \Sigma[E]$ we conclude that $X \rightarrow V \in \Sigma[E]^+_A$, which means that $V \subseteq X^+_\Sigma[E]$ holds as well. Consequently, $t(V) = t'(V)$, so $r$ satisfies $U \rightarrow V$. Also, We just showed that $X \rightarrow R$ is not implied by $\Sigma[E]$.

\[ \square \]

**Theorem 7.4.3 (E-BCNF).** Relation schema $R$ is in E-BCNF for the set $\Sigma$ of eUCs and eFDs if and only if $R$ is in BCNF for $\Sigma[E]$.

**Proof.** Let $R$ be in E-BCNF for a set of eUCs and eFDs over $R$. We show that $R$ is in BCNF for $\Sigma[E]$. For that purpose, let $X \rightarrow Y \in \Sigma[E]$ be a non-trivial FD over $R$. We need to show that $X \rightarrow R \in \Sigma[E]^+_A$ holds. If $Y = R$, then there is nothing to show. Otherwise, since $X \rightarrow Y \in \Sigma[E]$ and $Y \neq R$, the definition of $\Sigma[E]$ means that there is some non-trivial $E' : X \rightarrow Y \in \Sigma$ such that $E' \subseteq E$. Consequently, $E : X \rightarrow Y \in \Sigma^+_E$. Since $R$ is in E-BCNF for $\Sigma$, we also have $E : X \in \Sigma^+_E$. According to Lemma 7.4.2 that means $X \rightarrow R \in \Sigma[E]^+_A$, which is what we had to show.

Vice versa, let $R$ be in BCNF for $\Sigma[E]$. We need to show that $R$ is in E-BCNF for $\Sigma$. For that purpose, let $E : X \rightarrow Y \in \Sigma^+_E$ be a non-trivial eFD over $R$. We need to show that $E : X \in \Sigma^+_E$ holds. We first observe that $X \neq R$ as otherwise the given eFD would be trivial. Due to Theorem 7.3.3 and Theorem 7.3.4, $E : X \rightarrow Y \in \Sigma^+_E$ is equivalent to $X \rightarrow Y \in \Sigma[E]^+_A$. Since $R$ is in BCNF for $\Sigma[E]$, $X \rightarrow R \in \Sigma[E]^+_A$ holds, too. According to Lemma 7.4.2, $E : X \in \Sigma^+_E$ holds. \[ \square \]
Example 7.6 (Reduction of E-BCNF) For \( R = \{f, l, p, d\}, \Sigma = \{fld : fld, dpl : p \rightarrow l\} \) and \( E = dlp \), \( R \) is not in BCNF for \( \Sigma[E] = \{p \rightarrow l\} \).

7.4.3 E-RFNF

We can now characterize the semantic E-RFNF by the syntactic E-BCNF. Extending the well-known relational case, schemata in E-BCNF guarantee at application design time that there will never be a future instance that contains any E-redundant data value occurrence.

Theorem 7.4.4 (Semantics of E-BCNF). For all relation schemata \( R \), all attribute subsets \( E \subseteq R \), and all sets \( \Sigma \) of eUCs and eFDs over \( R \), \( R \) is in E-RFNF for \( \Sigma \) if and only if \( R \) is in E-BCNF for \( \Sigma \).

Proof. By Theorem 7.4.1, \( R \) is in E-RFNF for \( \Sigma \) if and only if \( R \) is in RFNF for \( \Sigma[E] \). However, the latter is equivalent to \( R \) being in BCNF for \( \Sigma[E] \). By Theorem 7.4.3, \( R \) being in BCNF for \( \Sigma[E] \) is equivalent to \( R \) being in E-BCNF for \( \Sigma \).

Example 7.7 (Semantics of E-BCNF) In our running example \( R = flpd \) and \( \Sigma = \{fld : fld, dpl : p \rightarrow l\} \), \( R \) is in dfl-RFNF for \( \Sigma \) since \( R \) is in BCNF for \( \Sigma[E] = \{fld \rightarrow fldp\} \).

7.4.4 Efficient Testing

Due to the cover-insensitivity of the E-BCNF condition, one may wonder about the efficiency of checking whether a given schema is in E-BCNF for a given set \( \Sigma \) of eUCs and eFDs. As in the classical case it suffices to check some eFDs in \( \Sigma \) instead of checking all eFDs in \( \Sigma[E] \).

Theorem 7.4.5 (Deciding E-BCNF). A relation schema \( R \) is in E-BCNF for a set \( \Sigma \) of eUCs and eFDs if and only if for every eFD \( E' : X \rightarrow Y \in \Sigma \) where \( E' \subseteq E \) and \( Y \not\subseteq X \), \( E : X \in \Sigma[E] \). Hence, deciding if a schema is in E-BCNF for \( \Sigma \) is quadratic in \( \Sigma \).
Proof. Since every eFD \( E' : X \rightarrow Y \in \Sigma \) where \( E' \subseteq E \) implies that \( E : X \rightarrow Y \in \Sigma^+_e \), the condition is necessary for \( R \) to be in \( E \)-BCNF for \( \Sigma \). It remains to show the opposite.

Assume that for every eFD \( E' : X \rightarrow Y \in \Sigma \) where \( E' \subseteq E \) and \( Y \not\subseteq X \), we have \( E : X \in \Sigma^+_e \). Next we show that \( R \) is in \( E \)-BCNF for \( \Sigma \). By Theorem 7.4.3, it suffices to show that \( R \) is in BCNF for \( \Sigma[E] \). That is, for every non-trivial FD \( X \rightarrow Y \in \Sigma[E] \) we need to show that \( X \rightarrow R \in \Sigma_{[E]}^+ \) holds. Let \( X \rightarrow Y \in \Sigma[E] \) be a non-trivial FD. By definition of \( \Sigma[E] \) it follows that i) \( Y = R \) or ii) there is some \( E' : X \rightarrow Y \in \Sigma \) such that \( E' \subseteq E \). If i) holds, then there remains nothing to show. Otherwise, ii) holds because our assumption implies that \( E : X \in \Sigma^+_e \). Lemma 7.4.2 shows that \( X \rightarrow R \in \Sigma_{[E]}^+ \).

Example 7.8 (Deciding \( E \)-BCNF) In our running example \( R = flpd \) and \( \Sigma = \{ fld : fld, dpl : p \rightarrow l \} \), \( R \) is in \( dfl \)-BCNF for \( \Sigma \) since there is no eFD \( E' : X \rightarrow Y \in \Sigma \) such that \( E' \subseteq E = dfl \).

7.4.5 \( E \)-3NF

We now introduce \( E \)-Third Normal Form (\( E \)-3NF) which ensures that all FDs can be enforced locally, without the need of joining relations to check for consistency of updates. We extend the classical concepts (see Chapter 2) to handle data completeness requirements as follows.

Definition 7.7 (\( E \)-minimal EUCs) For \( E \subseteq R \) and an eUC/eFD set \( \Sigma \), an eUC \( E : K \in \Sigma^+_e \) is \( E \)-minimal for \( \Sigma \) if and only if there is no eUC \( E : K' \in \Sigma^+_e \) such that \( K' \subset K \).

Definition 7.8 (\( E \)-prime Attribute) An attribute \( A \) is \( E \)-prime for \( \Sigma \) if and only if \( A \in K \) for some \( E \)-minimal eUC \( E' : K \in \Sigma^+_e \).
Definition 7.9 (E-3NF) A relation schema R is in E-3NF for \( \Sigma \) if and only if for every non-trivial eFD \( E' : X \rightarrow Y \in \Sigma_\emptyset^+ \) with \( E' \subseteq E \), \( E : X \in \Sigma_\emptyset^+ \) or every attribute in \( Y - X \) is E-prime.

Example 7.9 (E-3NF) In our running example \( R = flpd \) and \( \Sigma = \{ fld : fld, dpl : p \rightarrow l \} \), we have seen that dlp : flp is a dlp-minimal eUC for \( \Sigma \), the other dlp-minimal eUC being dlp : flp. That is, every attribute in \( R \) is dlp-prime. Hence, \( R \) is in dlp-3NF. However, \( R \) is not in dpl-3NF since for the eFD dpl : p \rightarrow l, l \notin \{ p \}, dlp : p \notin \Sigma_\emptyset^+ \), and l is not dpl-prime.

Similar to E-BCNF and BCNF, we can check that \( R \) is in E-3NF for \( \Sigma \) by testing that \( R \) is in 3NF for \( \Sigma \) [E].

Lemma 7.4.6 (E-prime Attribute). An attribute \( A \in R \) is E-prime for a given set \( \Sigma \) of eUCs and eFDs over \( R \) if and only if the attribute \( A \) is prime for \( \Sigma [E] \).

Proof. Lemma 7.4.2 shows that \( K \) is a minimal E-key for \( R \) for \( \Sigma \) if and only if \( K \) is a minimal key for \( R \) for \( \Sigma [E] \). Consequently, an attribute \( A \in R \) is E-prime for \( \Sigma \) if and only if \( A \) is prime for \( \Sigma [E] \).

Theorem 7.4.7 (E-3NF Reduction). For all relation schemata \( R \), all \( E \subseteq R \), and all sets \( \Sigma \) of eUCs and eFDs over \( R \), \( R \) is in E-3NF for \( \Sigma \) if and only if \( R \) is in 3NF for \( \Sigma [E] \).

Proof. Suppose \( R \) is in E-3NF for \( \Sigma \). We show that \( R \) is in 3NF for \( \Sigma [E] \). Let \( X \rightarrow Y \in \Sigma [E] \) be a non-trivial FD over \( R \). If \( Y = R \), then there remains nothing to show. Otherwise, there is some \( E' : X \rightarrow Y \in \Sigma \) such that \( E' \subseteq E \) and \( Y \nsubseteq X \). Hence, \( E : X \rightarrow Y \in \Sigma_\emptyset^+ \) and \( Y \nsubseteq X \). Since \( R \) is in 3NF for \( \Sigma \) it follows that i) \( E : X \in \Sigma_\emptyset^+ \), or ii) every attribute in \( Y - X \) is E-prime. Lemma 7.4.2 shows that \( X \rightarrow R \in \Sigma [E]_\emptyset^+ \). Furthermore, ii) implies that every attribute in \( Y - X \) is prime for \( \Sigma [E] \) by Lemma 7.4.6. That is, \( R \) is in 3NF for \( \Sigma [E] \).
Vice versa, suppose \( R \) be in 3NF for \( \Sigma[E] \). We show that \( R \) is in \( E \)-3NF for \( \Sigma \). Let \( E : X \rightarrow Y \in \Sigma^+_E \) be non-trivial. If \( Y = R \), then there remains nothing to show. So let \( Y \subset R \). From \( E : X \rightarrow Y \in \Sigma^+_E \) we conclude that \( X \rightarrow Y \in \Sigma[E]^+_\alpha \) is non-trivial. Since \( R \) is in 3NF for \( \Sigma[E] \), it follows that i) \( X \rightarrow R \in \Sigma[E]^+_\alpha \), or ii) every attribute in \( Y - X \) is prime for \( \Sigma[E] \). By Lemma 7.4.6, it follows that ii) implies every attribute \( Y - X \) is \( E \)-prime for \( \Sigma \). Consequently, \( R \) is in \( E \)-3NF for \( \Sigma \).

\[ \square \]

**Example 7.10 (E-3NF Reduction)** In our running example \( R = flpd \), \( \Sigma = \{fld : fld, dpl : p \rightarrow l\} \) and \( E = dflp \), \( \Sigma[E] = \{fld \rightarrow p, p \rightarrow l\} \). The two minimal eUCs are \( fld : fld \) and \( flp : flp \). As every attribute is prime, \( R \) is in 3NF for \( \Sigma[E] \).

Finally, \( E \)-3NF can be validated by checking the relevant conditions for just the input \( \Sigma \), rather than its closure \( \Sigma^+_E \).

**Theorem 7.4.8 (Deciding E-3NF).** Relation schema \( R \) is in \( E \)-3NF for a set \( \Sigma \) of eUCs and eFDs over \( R \) if and only if for every eFD \( E' : X \rightarrow Y \in \Sigma \) where \( E' \subseteq E \) and \( Y \not\subseteq X \), \( E : X \in \Sigma^+_E \) or every attribute in \( Y - X \) is \( E \)-prime.

**Proof.** Since every eFD \( E' : X \rightarrow Y \in \Sigma \) where \( E' \subseteq E \) implies that \( E : X \rightarrow Y \in \Sigma^+_E \), the condition is necessary for \( R \) to be in \( E \)-3NF for \( \Sigma \). It remains to show the opposite.

Assume that for every eFD \( E' : X \rightarrow Y \in \Sigma \) where \( E' \subseteq E \) and \( Y \not\subseteq X \), we have \( E : X \in \Sigma^+_E \) or every attribute in \( Y - X \) is \( E \)-prime. Next, we show that \( R \) is in \( E \)-BCNF for \( \Sigma \). By Theorem 7.4.7, it suffices to show that \( R \) is in 3NF for \( \Sigma[E] \). That is, for every non-trivial FD \( X \rightarrow Y \in \Sigma[E] \) we need to show that \( X \rightarrow R \in \Sigma^+_\alpha \) holds or every attribute in \( Y - X \) is prime for \( \Sigma[E] \).

Let \( X \rightarrow Y \in \Sigma[E] \) be a non-trivial FD. By definition of \( \Sigma[E] \) it follows that i) \( Y = R \), or ii) there is some \( E' : X \rightarrow Y \in \Sigma \) such that \( E' \subseteq E \). If i) holds, then there remains nothing to show. Otherwise, ii) holds and our assumption implies that \( E : X \in \Sigma^+_E \) or every attribute in \( Y - X \) is \( E \)-prime. Lemma 7.4.2
and Lemma 7.4.6 show that $X \rightarrow R \in \Sigma[E]_{\Sigma}$ or every attribute in $Y - X$ is prime for $\Sigma[E]$. Hence, $R$ is in 3NF for $\Sigma[E]$. This completes the proof. \qed

**Example 7.11 (Deciding E-3NF)** In our running example $R = flpd$, $\Sigma = \{fld : fld, dpl : p \rightarrow l\}$ and $E = dflp$, we can see that for $dpl : p \rightarrow l \in \Sigma$ the attribute $l$ is $E$-prime. By Theorem 7.4.8 this suffices to establish $R$ is in $E$-3NF for $\Sigma$. \qed

### 7.4.6 Hardness of Normal Form Criteria

As relational normalization represents the special case where $E = R$, checking normal form criteria is hard in general.

**Theorem 7.4.9** (NP-completeness). Deciding whether a sub-schema of a given schema is in $E$-BCNF for a given set $\Sigma$ of eUCs and eFDs is coNP-complete. Deciding whether a given schema is in $E$-3NF for a given set $\Sigma$ of eUCs and eFDs is NP-complete.

### 7.5 Tailoring Normalization

We now establish algorithms to design relational database schemata that are tailored to the completeness requirements of applications. For that purpose, we normalize a given schema $R$ for the given set $\Sigma$ of eUCs and eFDs. The completeness requirements are consolidated in an attribute subset $E \subseteq R$, expressing that the application only handles $E$-complete records. The choice of $E$ determines the set $\Sigma[E]$ of traditional FDs that are used to normalize $R$. For each $E$ we pursue i) lossless BCNF decompositions that are redundancy-free but potentially not dependency-preserving, and ii) lossless 3NF syntheses that are dependency-preserving but potentially not redundancy-free.
7.5.1 E-BCNF Decomposition

We recall terminology from relational databases. A decomposition of relation schema $R$ is a set $\mathcal{D} = \{R_1, \ldots, R_n\}$ of relation schemata such that $R_1 \cup \ldots \cup R_n = R$. For $R_j \subseteq R$ and FD set $\Sigma$ over $R$, $\Sigma[R_j] = \{X \rightarrow Y \mid X \rightarrow Y \in \Sigma^+_E \text{ and } X, Y \subseteq R\}$ denotes the projection of $\Sigma$ onto $R_j$. A decomposition $\mathcal{D}$ of $R$ with FD set $\Sigma$ is lossless if every relation $r$ over $R$ that satisfies $\Sigma$ is the join of its projections on the elements of $\mathcal{D}$, that is $r = \Join_{R_j \in \mathcal{D}} r[R_j]$. Here, $r[R_j] = \{t(R_j) \mid t \in r\}$. A BCNF decomposition of $R$ with FD set $\Sigma$ is a decomposition $\mathcal{D}$ of $R$ where every $R_j \in \mathcal{D}$ in BCNF for $\Sigma[R_j]$. Theorem 7.4.3 motivates a definition of an $E$-lossless BCNF decomposition.

**Definition 7.10 (E-lossless BCNF Decomposition)** An $E$-lossless BCNF decomposition of a schema $R$ for a set $\Sigma$ of eUCs and eFDs over $R$ is a lossless BCNF decomposition of $R$ for $\Sigma[E]$. ☐

Theorem 7.5.1 immediately follows Definition 7.10. It covers the classical decomposition theorem [95] as the special case where $E = R$. Data completeness-tailored normalization does not loose any records by following a hybrid decomposition approach. Given $E$, a relation is decomposed horizontally into its application-relevant part $r^E$ of $E$-total tuples, and its application-irrelevant part $r - r^E$ of tuples with missing data on $E$. Classical vertical decomposition can then be applied to $r^E$.

**Theorem 7.5.1 (E-lossless Join)**. Let $E : X \rightarrow Y$ be an eFD that satisfies the relation $r$ over relation schemata $R$. Then the set of $E$-total tuples of $r$ is the lossless join of its projection on $XY$ and $X(R - Y)$, that is, $r^E = r^E[XY] \bowtie r^E[X(R - Y)]$. Also, $r$ is the disjoint union of the set of $E$-total tuples of $r$, and the set of tuples of $r$ with missing data on some column in $E$, that is, $r = r^E \cup (r - r^E)$.

**Proof.** A relation $r$ satisfies $E : X \rightarrow Y$ if and only if $r^E$ satisfies the FD $X \rightarrow Y$. Consequently, the classical decomposition theorem [95] covers the first case.
that \( r^E = r^E[X,Y] \cong r[X(R - Y)] \). The second case is trivial. 

Hence, an \( E \)-lossless BCNF decomposition for a set \( \Sigma \) of eUCs and eFDs can simply be obtained by a classical lossless BCNF decomposition for the \((E, FD)\)-reduct \( \Sigma[E] \) of \( \Sigma \).

<table>
<thead>
<tr>
<th>Problem: E-BCNF Decomposition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input: Relation schema ( R )</td>
</tr>
<tr>
<td>Set ( \Sigma ) of eUCs and eFDs over ( R )</td>
</tr>
<tr>
<td>Attribute subset ( E \subseteq R )</td>
</tr>
<tr>
<td>Output: ( E )-lossless BCNF decomposition of ( R ) for ( \Sigma )</td>
</tr>
<tr>
<td>Method: Perform a lossless BCNF decomposition of ( R ) for ( \Sigma[E] )</td>
</tr>
</tbody>
</table>

We illustrate the decomposition on our running example.

**Example 7.12 (E-lossless Decomposition)** In our running example \( R = flpd \) and \( \Sigma = \{ fld : fld, dpl : p \rightarrow l \} \), \( R \) is not in \( dflp \)-BCNF for \( \Sigma \). In fact, \( R \) is not in BCNF for \( \Sigma[dflp] = \{ dfl \rightarrow p, p \rightarrow l \} \). A BCNF decomposition yields \( R_1 = l p \) with \( \Sigma_1 = \{ p \rightarrow l \} \) and \( R_2 = dfp \) with \( \Sigma_2 = \emptyset \). For the relation \( r \) from Table 7.1, the projection of \( r^E \) on the decomposed schemata is as follows.

<table>
<thead>
<tr>
<th>last_name</th>
<th>phone</th>
<th>first_name</th>
<th>phone</th>
<th>day_register</th>
</tr>
</thead>
<tbody>
<tr>
<td>Futrell</td>
<td>252 398 3716</td>
<td>Sallie</td>
<td>252 398 3716</td>
<td>10/21/2008</td>
</tr>
<tr>
<td>Herbert</td>
<td>252 398 3716</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

All \( E \)-redundant data value occurrences from \( r \) have been eliminated. However, the eUC \( df l : df l \) was not preserved.

Recall that a decomposition \( D \) of schema \( R \) with FD set \( \Sigma \) is dependency-preserving whenever \( \Sigma^+_c = (\bigcup_{R_j \in D} \Sigma[R_j])^+ \).

**Definition 7.11 (E-dependency-preserving Decomposition)** An \( E \)-dependency-preserving decomposition of a schema \( R \) for the eUC/eFD set \( \Sigma \) is a dependency-preserving decomposition of \( R \) for \( \Sigma[E] \).
7.5.2 \( E \)-3NF Synthesis

3NF synthesis guarantees dependency-preservation, but may exhibit data value redundancy caused by FDs. It was shown recently that 3NF exhibits minimal levels of data redundancy when achieving dependency-preservation [8], [63]. Hence, we will equip our new framework with an appropriate 3NF synthesis strategy. Recall that a 3NF decomposition of a relation schema \( R \) for an FD set \( \Sigma \) is a decomposition \( D \) of \( R \) where every \( R_j \in D \) is in 3NF for \( \Sigma[R_j] \). Theorem 7.4.7 motivates the following definition.

**Definition 7.12 (\( E \)-3NF Decomposition)** An \( E \)-dependency-preserving, \( E \)-lossless 3NF decomposition of a schema \( R \) for the set \( \Sigma \) of eUCs and eFDs is a dependency-preserving, lossless 3NF decomposition of \( R \) for \( \Sigma[E] \).

Following Theorem 7.5.1, an \( E \)-dependency-preserving, \( E \)-lossless 3NF synthesis for a set \( \Sigma \) of eUCs and eFDs can simply be obtained by a classical dependency-preserving-lossless 3NF synthesis for the \( (E, FD) \)-reduct \( \Sigma[E] \) of \( \Sigma \).

<table>
<thead>
<tr>
<th>Problem:</th>
<th>( E )-3NF Synthesis</th>
</tr>
</thead>
</table>
| Input: | Relation schema \( R \) 
Set \( \Sigma \) of eUCs and eFDs over \( R \) 
Attribute subset \( E \subseteq R \) |
| Output: | \( E \)-dependency-preserving, \( E \)-lossless 3NF decomposition of \( R \) for \( \Sigma[E] \) |
| Method: | Perform a dependency-preserving, lossless 3NF synthesis of \( R \) for \( \Sigma[E] \) |

We illustrate the synthesis on our running example.

**Example 7.13 (\( E \)-3NF Synthesis)** In our running example \( R = flpd \) and \( \Sigma = \{fld : fld, dpl : p \rightarrow l\} \), \( R \) is indeed in \( dflp \)-3NF for \( \Sigma \). For \( E = dlp \), however, \( R \) is not in \( E \)-3NF for \( \Sigma \). In fact, \( R \) is not in 3NF for \( \Sigma[E] = \{p \rightarrow l\} \). A 3NF
synthesis yields $R_1 = \{p\} \rightarrow l$ and to ensure $E$-losslessness we add $R_2 = d_{fp}$ with $\Sigma_2 = \emptyset$. For relation $r$ from Table 7.1, the projection of $r^E$ onto the decomposition schema is as follows.

<table>
<thead>
<tr>
<th>first_name</th>
<th>phone</th>
<th>day_register</th>
</tr>
</thead>
<tbody>
<tr>
<td>Futrell</td>
<td>252 398 3716</td>
<td>11/21/2008</td>
</tr>
<tr>
<td>Sallie</td>
<td>252 398 3716</td>
<td>10/21/2008</td>
</tr>
<tr>
<td>Herbert</td>
<td>252 398 3716</td>
<td>10/21/2008</td>
</tr>
</tbody>
</table>

All $E$-redundant data value occurrences from $r$ have been eliminated. In fact, all decompositions are in $BCNF$ for their projected sets of FDs. Note the difference to the decomposition in Example 7.12. Due to looser data-completeness requirements, the first tuple is included in the application-relevant part $r^E$, and thus in the decomposition.

We have tailored the entire classical schemata design framework to data-completeness requirements of applications. We allow data stewards to declare these requirements as an extension to the familiar concept of a functional dependency. The results show that extensions of the familiar $BCNF$ (3NF) achieve an elimination (minimization across dependency-preserving decompositions) of data values that may occur redundantly in tuples that meet the completeness requirements. As an optimal result for database practice, schemata can be automatically transformed into these normal forms by applying classical normalization algorithms to a set of classical FDs that emerge from the set of extended FDs and the data-completeness requirements. The next section illustrates what our framework achieves when applied to real-world schemata and data.
Table 7.4: Time of computing redundant data value occurrences in benchmarks

<table>
<thead>
<tr>
<th>Dataset</th>
<th>#Complete values</th>
<th>#Redundancy</th>
<th>%Redundancy</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>horse</td>
<td>6795</td>
<td>4775</td>
<td>70.27</td>
<td>8.075</td>
</tr>
<tr>
<td>bridges</td>
<td>1327</td>
<td>411</td>
<td>30.97</td>
<td>0.002</td>
</tr>
<tr>
<td>hepatitis</td>
<td>2933</td>
<td>1695</td>
<td>57.79</td>
<td>0.179</td>
</tr>
<tr>
<td>cancer</td>
<td>7585</td>
<td>712</td>
<td>9.39</td>
<td>0.005</td>
</tr>
<tr>
<td>echo</td>
<td>1584</td>
<td>489</td>
<td>30.87</td>
<td>0.002</td>
</tr>
<tr>
<td>plista</td>
<td>39431</td>
<td>28827</td>
<td>73.11</td>
<td>18.415</td>
</tr>
<tr>
<td>flight</td>
<td>57062</td>
<td>48414</td>
<td>84.84</td>
<td>86.585</td>
</tr>
<tr>
<td>ncvoter1k</td>
<td>16137</td>
<td>3170</td>
<td>19.64</td>
<td>0.047</td>
</tr>
<tr>
<td>china</td>
<td>4313980</td>
<td>2131677</td>
<td>49.41</td>
<td>412.867</td>
</tr>
<tr>
<td>uniprot512k</td>
<td>11600704</td>
<td>1413038</td>
<td>12.18</td>
<td>1777.245</td>
</tr>
<tr>
<td>diabetic</td>
<td>1017738</td>
<td>543935</td>
<td>53.45</td>
<td>3273.183</td>
</tr>
</tbody>
</table>

7.6 Experiments

We report on experiments with our framework using real-world benchmark schemata and data. We provide insight on how many $E$-redundant data values occur in the data, rank the relevance of eFDs by how many data redundancies they cause, show how often schemata satisfy a normal form condition, how much redundancy $E$-3NF permits, how many dependencies $E$-BCNF preserves, and how large decomposed schemata become. We consider the times of computations, and suggest how data stewards can use our ranking of eFDs, using example our two applications from the introduction. All our experiments are run on an Intel Xeon 3.6GHz, 256GB RAM, Windows 10 Dell workstation.

7.6.1 $E$-Redundancy and EFD Ranking

Table 7.4 lists for each incomplete benchmark the number of complete data values (#Complete values), the number of redundant data values (#Redundancy), the percentage of redundant data values (%Redundancy), and the time in seconds to compute all the data redundancies given the dataset and given the canonical cover of the eUCs and eFDs that hold on the dataset.
The sheer number and percentage of redundant data values clearly motivate our research, and the time taken to compute them shows that this insightful analysis is efficient on even large datasets with large numbers of constraints. Of course, if a team of domain experts selects the meaningful eUCs and eFDs for an application, then the redundant occurrences will likely to be fewer and can be computed more efficiently.

In order to guide data stewards in their selection of meaningful eFDs from the discovered ones, we rank the relevance of an eFD by the number of redundant data value occurrences it causes. Figure 7.3 shows the number of eFDs in a canonical cover that cause not more than a given number of redundant data value occurrences in the dataset. The labels on the x-axis indicate the maximum values for 0, 2.5, 5, 10, 15, 20, 40, 60, 80, 100 percent of the maximum redundant data value occurrences that some existing eFD causes. The figure illustrates clearly that most eFDs cause few data redundancies, which makes it possible for data stewards to focus their attention to select few eFDs of higher relevance.

We further apply the rankings of eFDs by their data redundancies to various row-fragments of the *ncvoter* dataset. The sizes of *ncvoter* datasets vary on numbers of rows from $2^i$ thousand tuples for $i = 0, \ldots, 10$. Figure 7.4 shows the ranking percentiles of eFDs discovered on those fragments, as well as the times in seconds to compute all the $E$-redundant data value occurrences.

### 7.6.2 Quality of Decompositions

For the following experiments we created inputs as outlined next. For each fixed size $|E|$ of the completeness requirements, we created different sets of eUCs and eFDs by picking up to 1,000 unique attribute sets $E$ of the fixed size, and then selecting all eUCs and eFDs that hold on the dataset and whose embeddings are subsets of $E$. Figure 7.5 shows for each dataset and each size of $E$, the percentage of all input sets that are in $E$-BCNF and $E$-3NF, respectively.
Figure 7.3: EFD relevance distributions of benchmark datasets
7.6. Experiments

Figure 7.4: EFD relevance distributions of *ncvoter* datasets varying on different numbers of rows.
Figure 7.5: Average percentage of schemata in E-3NF and E-BCNf by given size of E
Figure 7.6 shows the average percentages of i) the $E$-complete data values that are redundant (blue line), ii) those after $E$-3NF decomposition (orange line) and iii) eliminated redundancies after $E$-3NF synthesis (yellow line) all plotted against the left-hand-side vertical axis, and iv) dependencies preserved during $E$-BCNF decomposition (red dotted line) plotted against the right-hand-side vertical axis. In general, there is no control about the number of $E$-redundant data values that an $E$-3NF decomposition must tolerate to preserve all relevant dependencies. Vice versa, there is no control on how many relevant dependencies will be lost to eliminate all $E$-redundant data values during $E$-BCNF decomposition. For instance, $E$-3NF may duplicate non-trivial eFDs across various schemata, causing an actual blow-up of the $E$-redundancies, see diabetic and weather.

Figure 7.7 reports the same criteria about data redundancies as in Figure 7.6 but shows for the different row-fragments of ncvoter. It is very interesting to see how stable the results are across the vastly varying numbers of rows in the fragments.

### 7.6.3 Size and Time of Decompositions

Figure 7.8 illustrates the impact of the size $|E|$ on the cardinality of the decompositions, that is, their total number of attributes. Boldly speaking, the larger the decompositions the more updates (less redundancy) and the less queries (more joins required) will be efficient.

### 7.6.4 Qualitative Analysis

Finally, we comment on the use of our rankings in identifying relevant eFDs for normalization or identifying dirty data. Figure 7.9 shows the distribution of the various eFDs according to the percentiles of redundant data value occurrences they cause. The two applications illustrate already that with growing $|E|$ usually more eFDs need to be considered. Here, our ranking offers a convenient measure of relevance for data stewards.
Figure 7.6: Elimination of $E$-redundancy by $E$-3NF (vertical left-hand-side), and $E$-preservation by $E$-BCNF (vertical right-hand-side)
FIGURE 7.7: Elimination of E-redundancy by E-3NF (vertical LHS), and E-preservation by E-BCNF (vertical RHS) by row-fragments of \textit{necvoter}
FIGURE 7.8: Average cardinality of decompositions and time (s) taken to compute them.
A view that might be particularly useful for data stewards is to fix a column in $E$, and list the minimal LHSs that functionally determine that column, ranked by the relevance of the corresponding eFD. This is illustrated on Figures 7.10 and 7.11, where we list all minimal LHSs for the fixed columns $city$ and $zip\_code$ for our two applications with requirements $E_1$ and $E_2$, respectively.

Finally, a data steward can view the records in which the redundant data values actually occur. This is to help the steward decide whether the eFD is actually semantically meaningful for the application, holds just accidentally, or to identify some records with dirty data. Figure 7.12 shows some tuples with $E_1$-redundant data value occurrences.

An inspection of these records reveals some dirty data: i) Hazel and Homer Hargis live at the same street address, and their phone numbers are different, and ii) Vivian and John Etheridge share the same phone number, but their street address is different. For i) the inconsistency can easily be resolved by giving the full phone number, while for ii) it is more likely that
Figure 7.11: Relevance distribution of eFDs in ncvoter1k for application $E_2$

Figure 7.12: $E_1$-redundancies caused by eFD `full_phone_num: last_name, city, register_date → zip_code`

Vivian indicated the correct street number.

Our experiments illustrate on benchmark schemata and data that eFDs provide effective declarative means to capture and reason about redundant data value occurrences that are fit for application requirements. Our ranking guides data stewards in their selection of eFDs that are relevant for normalization purposes given the application requirements. Our normalization strategies result in a wide spectrum of schemata with clear achievements in terms of the elimination of pertinent data redundancies or the preservation of pertinent dependencies, accommodating tradeoffs between update and query efficiency.

7.7 Summary

From the perspective of data quality, relational database schema design only addresses the integrity dimension. Hence, data stored in the application
schemata are only fit for purpose from a data integrity point of view. We have embarked on a new line of research whose aim is to develop a general schema design framework that accommodates also other data quality dimensions. Based on the novel class of embedded functional dependencies, we have fully generalized the normalization framework around BCNF and 3NF to accommodate data completeness requirements of applications. Extensive experiments on real-world benchmark schemata and data exemplified the effectiveness of our framework, the efficiency of our algorithms, and the achievements of our new normal form proposals. In particular, we illustrated the impact of the completeness requirements on trade-offs between data redundancy elimination and dependency-preservation.
Chapter 8

Discovering Embedded Functional Dependencies

In Chapter 7 we introduced embedded functional dependencies (eFDs) that generalize classical functional dependencies with additional constraints on the completeness dimension of data. In the final technical chapter of this thesis, we will address the discovery problem of eFDs. Indeed, establishing syntactic and semantic profiles with the help of eFDs provides new insight into data. For example, eFDs that exhibit high levels of data redundancy are more likely to be meaningful to the underlying application domain. Driving normalization by decomposing along those eFDs can eliminate data redundancy. Surprisingly, the number of redundant data values an FD causes had not been utilized for ranking discovered FDs prior to our work. For these reasons, building eFD-based data profiles and discovering eFDs from existing data is an important computational problem. As a whole, Chapter 8 starts realizing a promising framework in which discovered eFDs drive an automated schema design process for incomplete data.

After a motivation for the discovery of eFDs and an overview of our results in Section 8.1, we establish the complexity of the decision variant associated with the discovery problem for eFDs in Section 8.2. Section 8.3 introduces the data structure that we will use in our discovery algorithms. Section 8.4, Section 8.5, and Section 8.6 will introduce the first row-efficient,
column-efficient, and hybrid approaches towards the discovery of eFDs, respectively. Section 8.7 details an analysis of our experiments regarding these discovery algorithms. Semantic profiles of eFDs in the form of Armstrong samples are developed and experimentally analyzed in Section 8.8. A brief summary in Section 8.9 concludes this chapter.

8.1 Introduction

Data profiling is the process that computes meta-data about a given dataset [1]. It is of great importance to many applications including data integration, data quality, data cleaning, query optimization, and data repository design. One of the most important tasks in data profiling is the discovery of data dependencies that hold on the given dataset. However, the study of profiling data with incomplete information has not received dedicated attention in recent decades. Our novel notion of embedded data dependencies separate completeness from integrity concerns. More importantly, they lift many applications from complete to incomplete data. In this chapter, we focus on the discovery problem of embedded functional dependencies. Recall that the implication problem and design theory of eFDs and eUCs was addressed in Chapter 7.

An embedded functional dependency (eFD) is a statement of the form of $E : X \rightarrow Y$ where $X, Y \subseteq E$ and $X, Y$ are attribute sets from relation schema $R$. It states that the combination of values on $X$ uniquely determines each value in $Y$ for the subset relation within scope $E$. Unlike classical functional dependencies assuming no missing information (a.k.a null markers) in the given data, eFD discoveries not only discover functional dependencies but also divide them into different complete fragments. This also can be considered as a structured view of functional dependencies. Providing such view has two main advantages of profiling incomplete data. On one hand, eFDs prevent null markers from being misinterpreted and keep truthful data dependencies
which are supported by complete data. On the other hand, discovering eFDs can also act as an entry point for the discovery problems of other classes of FDs over incomplete data. Embedded FDs do not conflict with other classes of FDs that interpret null markers, because any interpretation of null markers is ignored by eFDs.

Here, we showcase an example of real-world data profiled by eFDs. The North Carolina Voters (ncvoter) dataset contains personal information about voters from North Carolina, United States [87]. Many applications can be built upon the data to help make important decisions on running electoral campaigns in the state of North Carolina. In fact, the dataset is massive and lots of important information such as phone numbers and addresses are missing from the collected records. Without any initial examination, it is impossible to know what questions can be answered by the data. Profiling the data with the use of classical data dependencies such as functional dependencies does not provide much help in building applications, because the semantics of these FDs is usually dependent on some fixed interpretation of null markers. On the other hand, embedded data dependencies such as eFDs can provide insight into incomplete data using a generic multidimensional view of different complete fragments. Figure 8.1 visualizes a snippet of eFDs discovered from the ncvoter dataset as a multidimensional view. Given a set of completeness requirements by some application such as full_phone_num, birth_place, city, zip_code, street_address, eFDs can horizontally slice the requirements into several non-inclusive dimensions, such as {full_phone_num, birth_place, city, zip_code} and {full_phone_num, birth_place, street_address}. Each of the dimensions can be sliced even further into subdimensions, for instance {full_phone_num, birth_place}, {full_phone_num, city}, and {full_phone_num, zip_code}. Each dimension is measured by its completeness and data redundancy. By combining different dimensions, FDs can be explored. Hence, building eFD profiles does not only provide a rich context for analysing incomplete data but can also infer insight on data quality.
Another motivation for discovering eFDs is their use for eliminating data redundancy from incomplete data. In Figure 8.2, the example table on the top contains several redundant data value occurrences in the *city* column, such as *burlington*. In fact, no classical functional dependencies can be identified to perform lossless-join decompositions to reduce data redundancy. Instead, if the eFD \{\textit{Phone}, \textit{City}, \textit{Zip}\} : \textit{Zip} → \textit{City} holds, the subrelation that is complete on \{\textit{Phone}, \textit{City}, \textit{Zip}\} can be decomposed into \{\textit{City}, \textit{Zip}\} and \{\textit{Id}, \textit{Address}, \textit{Zip}, \textit{Phone}\}, effectively reducing data redundancy in column \textit{City}.

Note that the example shown in Figure 8.2 is only a small snippet, and the original dataset actually contains 20 columns and over 1 million rows. The discovery of eFDs and their ranking by the number of redundant data values they cause appears to provide necessary computational support for realizing such decompositions in practice.

Our main contributions in this chapter are summarized as follows:

1. We show that the decision variant of eFD discovery is \textit{NP}- and \textit{W}[2]-complete.
2. We introduce the novel data structure of eFD-trees that enables us to store and search eFDs efficiently.

3. We introduce the first row-efficient algorithm, column-efficient algorithm, and hybrid algorithm for eFD discovery.

4. We introduce Armstrong samples for eFDs, and give algorithmic characterizations for computing Armstrong samples of eFDs.

5. We extensively demonstrate the practical usefulness of our algorithms by analyzing their performance on real-world datasets in terms of runtime and memory consumption.

6. We compute Armstrong samples of real-world benchmark datasets.

### 8.2 Computational Complexity

Before introducing any solutions to the eFD discovery problem, we first discuss the computational complexity of the decision problem EFD, defined as follows.

<table>
<thead>
<tr>
<th>Problem:</th>
<th>EFD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input:</td>
<td>relation $r$ over schema $R$ positive integer $k$</td>
</tr>
<tr>
<td>Output:</td>
<td>yes, if there is some $E : X \rightarrow A$ satisfied by $r$ where $X \subseteq E \subseteq R$, $A \in E - X$ and $</td>
</tr>
</tbody>
</table>
Chapter 8. Discovering Embedded Functional Dependencies

Note that we use the cardinality $|E|$ to represent the size of eFD $E : X \rightarrow A$ because $XA \subseteq E$. We can see that discovering eFDs is at least as hard as discovering FDs in complete relations, as eFDs are a generalization of FDs. It is well-known that the decision problem FD, defined as follows, is $NP$-complete [69]. By reducing FD to eFD, we can establish the $NP$-completeness for eFD, too.

<table>
<thead>
<tr>
<th>Problem:</th>
<th>FD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input:</td>
<td>relation $r$ over schema $R$</td>
</tr>
<tr>
<td></td>
<td>positive integer $k$</td>
</tr>
<tr>
<td>Output:</td>
<td>yes, if there is some $X \rightarrow A$ satisfied by $r$</td>
</tr>
<tr>
<td></td>
<td>where $X \subseteq R$, $A \in R - X$ and $</td>
</tr>
<tr>
<td></td>
<td>no, otherwise</td>
</tr>
</tbody>
</table>

**Theorem 8.2.1** (NP-completeness of eFD). The problem eFD is $NP$-complete.

**Proof.** eFD is in $NP$ because we can guess $E : X \rightarrow A$ with $|E| \leq k$ and verify in polynomial time using Algorithm 23 (see Section 8.4). For the $NP$-hardness, we reduce FD to eFD. Take an instance $(r, k)$ of FD where $r$ is a complete relation over relation schema $R$, and $k$ is positive integer. Let $(r', k')$ be the instance of eFD where $r = r'$ and $k = k'$. Now it follows that a non-trivial FD $X \rightarrow A$ where $|XA| \leq k$ is satisfied by $r$ if and only if the eFD $X : X \rightarrow A$ is satisfied by $r'$. □

In recent research [20], FD was shown to be $W[2]$-complete in the size of an FD. As we can show that FD and eFD are FPT-equivalent, it follows that eFD is also an intractable problem even if the size of eFDs are fixed.

**Theorem 8.2.2** (Fixed-parameter Complexity). The problem eFD is $W[2]$-complete.

**Proof.** We show that eFD and FD are equivalent under FPT-reductions. The result then follows from the $W[2]$-completeness of FD in [20]. For $FD \leq_{FPT} eFD$ the PTIME reduction is the same as the construction used for Theorem 8.2.1 since parameter $k'$ only depends on $k$. It remains to show that
8.3. EFD-trees

EFD≤_{FPT}FD holds. Take an instance \((r, k)\) of eFD. We transform \((r, k)\) into an instance \((r', k')\) by defining \(r'\) as the result of replacing null marker occurrences in \(r\) with unique column values in \(r'\), and defining \(k'\) to be \(k\). Clearly, this transformation is FPT. Now we claim that there is some eFD \(E : X \rightarrow A\) satisfied by \(r\) if and only if there is some FD \(E - A \rightarrow A\) satisfied by \(r'\) where \(|E| \leq k\). If there is an eFD \(E : X \rightarrow A\) where \(|E| \leq k\) that is satisfied by \(r\), then \(r^E\) satisfies \(X \rightarrow A\) furthermore \(r^E \models E \rightarrow A\); for any other tuples they satisfy \(E \rightarrow A\) trivially in \(r'\) because for any tuple \(t, t' \notin r^E\) there is a null maker in \(t(E)\) and \(t'(E)\) hence \(t(E) \neq t'(E)\) with respect to \(r'\). If there is an FD \(E - A \rightarrow A\) satisfied by \(r'\) where \(A \in E\) and \(|E| \leq k\), then there is \(E : X \rightarrow A\) satisfied by \(r\) where \(X \subseteq E - A\) since \(E : E - A \rightarrow A\) must be satisfied by \(r\). This concludes the proof.

Despite the likely intractability of the eFD discovery problem, it is still possible to develop efficient algorithms to process real-world datasets. By utilizing efficient data structures and generalized heuristics, we are able to design and implement a series of efficient algorithms as a toolkit to solve the eFD discovery problem on real-world data. In the following sections, we develop our eFD discovery algorithms.

8.3 EFD-trees

First of all, we recall the definition of eFDs (Definition 7.1). For the convenience of computation and discussions, we define the concise form of eFD \(E : X \rightarrow Y\) to be the expression \(E - XY : X \rightarrow Y\). On one hand, the concise representation of an eFD is more reader-friendly since the embedding attributes are less verbose and only capture additional completeness requirements required by an eFD. On the other hand, eliminating duplicate attributes in the extension is more efficient while searching and updating eFDs that are maintained by tree-like data structures.
Given complete relations, discovering data dependencies such as functional dependencies, keys often requires a special data structure to consistently store and update a set of intermediate results. These data structures normally serve with efficient functionalities such as accessing, searching and eliminating redundancies for the underlying data dependencies. Particularly, data structures such as antecedent trees, FD-trees, and eUC-trees have significant impact on improving runtime performance on discovering keys, FDs and eUCs respectively. Next, we introduce a novel data structure eFD-tree that is specialized for eFD discoveries.

**Definition 8.1 (EFD-tree)** Let R be a relation schema with a total order on its attributes. An eFD-tree is a tree with a unique l(left-hand-side)-root, multiple e(embedding)-roots, e(embedding)-nodes, l(left-hand-side)-nodes and satisfy the following properties:

1. Every node, except root nodes, represents an attribute of R;
2. All children of an e-root are e-nodes;
3. E-nodes only have e-node children;
4. Efd-nodes are the e-nodes labeled with RHS attributes of an eFD;
5. L-nodes and the l-root can have l-nodes children and at most one e-root child;
6. All child nodes, except e-roots, represent larger attributes than their non-root parents;
7. Each traversal of the tree from the l-root to an efd-node represents an eFD in the concise form.

**Example 8.1 (EFD-tree)** Figure 8.3 shows three different choices to store the set eFDs \(\{AB : A \rightarrow B, AC : A \rightarrow C\}\). Tree (a) is directly adapted from eUC-trees which uses embedding attributes as the prefix of a path. Tree (b) reverses the
construction of (a) so that LHS attributes become the prefix of a path. Tree (c) is our proposed eFD-tree in Definition 8.1 which follows the construction of (b) but only store eFDs in concise forms i.e. $\emptyset : A \rightarrow B$, $\emptyset : A \rightarrow C$.

Unlike eUC-trees (see Definition 6.2), eFD-trees only store eFDs in concise forms and use LHS attributes as the prefix of a path instead of embedding attributes. In this way, an eFD-tree is able to use one path to represent as many eFDs as possible, which dramatically improves the performance of searching and updating processes. As shown by tree (c) in Example 8.1, the eFD-tree can easily converge different eFDs into one single path without any ambiguities.

During an eFD discovery, newly discovered eFDs could be redundant to some existing eFDs which are more general e.g. if eFD $AB : A \rightarrow B$ is satisfied, then $ABC : A \rightarrow B$ is also satisfied. So, the main purpose of an eFD-tree is to provide functionalities of searching general eFDs. Algorithm 20 demonstrates how to determine if an eFD is redundant.

In the following sections, we introduce our eFD discovery algorithms. In particular, given an incomplete relation, the algorithms discover the canonical cover of eFDs satisfied by the relation.
Algorithm 20  Check Redundancy of EFD

1: **Input:** eFD $E : X \rightarrow A$, l-root node $l\_root$ of the eFD-tree over a set of eFDs $\Sigma$
2: **Output:** true there is $E' : X' \rightarrow A \in \Sigma$ where $E' \subseteq E$ and $X' \subseteq X$
3: return $is\_redundant(l\_root, E, X, A)$
4: function $is\_redundant(node, E, X, A)$
5: if node is eFD-node then
6: Let RHS(node) be the RHS attributes labeled on node
7: if $A \in RHS(node)$ then
8: return true
9: if node is e-node then
10: for each child e-node child of node do
11: Let $A$ be the attribute of child
12: if $A \in E$ and $is\_redundant(child, E, X, A)$ then
13: return true
14: if node is l-node then
15: if node has an e-root node $e\_root$ then
16: if $is\_redundant(e\_root, E, X, A)$ then
17: return true
18: for each child l-node child of node do
19: Let $A$ be the attribute of child
20: if $A \in X$ and $is\_redundant(child, E, X, A)$ then
21: return true
22: return true

Definition 8.2 (Canonical Cover of EFDs) Let $\Sigma$ be a set of eFDs over relation schema $R$. The canonical cover of $\Sigma$ is the set of non-trivial eFDs $\{ E : X \rightarrow A \mid \Sigma \models (E : X \rightarrow A), \forall B \in E - XA : \Sigma \not\models (E - B : X \rightarrow A), \forall B \in X : \Sigma \not\models (E : X - B \rightarrow A) \}$.

8.4  Row-efficient Discovery

In this section, we introduce the the row-efficient (column-based) discovery algorithm for eFDs, whose runtime mainly depends on the size of a relation schema.

In FD discoveries, a row-efficient algorithm models LHSs of FDs as an attribute lattice (as shown on the left of Figure 8.4) and uses the FD validation algorithm to find the satisfiable RHS for a given LHS. However, the classical approach such as [52] cannot be directly adapted to discover eFDs. To discover eFDs in a row-efficient manner, there are three main challenges to overcome. Firstly, the search space of eFDs includes both embeddings and LHSs, which is much larger and more complex than a single dimensional
attribute lattice. Secondly, discovering eFDs needs to search through all possible combinations of embeddings and LHSs. An efficient strategy must be devised to traverse and prune the search space of eFDs. Thirdly, eliminating eFDs with redundant embeddings involves examining eFDs discovered at all lower levels, unlike the row-efficient discovery algorithms for classical FDs which only has to inspect valid FDs from one level lower.

As shown on the right of Figure 8.4, the attribute lattice of embedding attributes is nested in the attribute lattice of LHSs. So, we propose a nested traversal strategy for the row-efficient discovery of eFDs. The nested traversal starts with a \textit{l-traversal} which will examine the attribute lattice of LHSs from lower to higher levels. For each LHS, an \textit{e-traversal} will be used to discover all possible embeddings for a given pair of LHS and RHS. During an e-traversal, eFDs are examined for validity and redundancy. If an eFD is valid and non-redundant, then it will be stored into an eFD-tree. Since e-traversals are nested in an l-traversal, the classical pruning strategy is no longer valid in general i.e. the validity of $E : X \rightarrow A$ does not imply the validity of $E' : XB \rightarrow A$ since $E$ is not necessary a subset of $E'$, which means finding satisfiable embeddings for a LHS may not reduce the attribute lattice of LHSs.
Instead, to improve the runtime performance of the row-efficient eFD discovery we can actually avoid unnecessary e-traversals and reduce attribute lattices of embeddings using the following propositions.

**Proposition 8.4.1** (Minimal Embedding of EFD). Let $\Sigma$ be the canonical cover of eFDs of the relation $r$ over $R$. $R(\bot)$ is the set \{$A \in R \mid \exists t \in r : t(A) = \bot$\}. For any eFD $E : X \rightarrow A \in \Sigma$, $E \setminus XA \subseteq R(\bot)$.

**Proof.** Let $r$ be a relation over relation schema $R$ and $R(\bot) = \{A \in R \mid \exists t \in r : t(A) = \bot\}$. Take any $E : X \rightarrow A \in \Sigma$ where $XA \subseteq E$ and $B \in E \setminus XA$. Assume $B \not\in R(\bot)$. By the definition of eFDs, $t_1(X) = t_2(X)$ implies $t_1(A) = t_2(A)$ for any $t_1, t_2 \in r^E$. If $B \not\in R(\bot)$, then $t(B) \neq \bot$ for all $t \in r$. In other words, $r^B = r$. Furthermore, we can derive $r^{E-B} = r^E \cup (r^E \cap r^B)$. Namely, $r^{E-B} = r^E$. However, $r \vdash E - B : X \rightarrow A$ and $E - B : X \rightarrow A$ must be in the canonical cover $\Sigma$ rather than $E : X \rightarrow A$, which draws a contradiction. Therefore, $B$ must be in $R(\bot)$. \hfill \Box

**Proposition 8.4.2** (Existence of Embedding). Let $r$ be a relation over relation schema $R$. For any $X \subseteq R$ and $A \in R - X$, there is $E \subseteq R$ where $XA \subseteq E$ such that $r \vdash E : X \rightarrow A$ if and only if $t_1(A) = t_2(A)$ for all $t_1, t_2 \in S \cap r^R$ and for all $S \in \pi_X$.

**Proof.** Let $r$ be a relation over relation schema $R$ an. Take any $X \subseteq R$ and $A \in R - X$. $\pi_X$ is the stripped partition of $X$ over $r$.

**Case $\Rightarrow$:** Suppose there exists $E \subseteq R$ where $XA \subseteq E$ such that $r \vdash E : X \rightarrow A$. Take any $S \in \pi_X$. Assume there exists $t_1, t_2 \in S \cap r^R$ where $t_1(A) \neq t_2(A)$. However, $t_1(X) = t_2(X)$ and $t_1, t_2 \in r^R \subseteq r^E$, which contradicts $r \vdash E : X \rightarrow A$.

**Case $\Leftarrow$:** Suppose $t_1(A) = t_2(A)$ for all $t_1, t_2 \in S \cap r^R$ and for all $S \in \pi_X$. Therefore, $t_1(X) = t_2(X)$ implies $t_1(A) = t_2(A)$ if $t_1, t_2 \in r^R$. In other words, $r \vdash R : X \rightarrow A$. \hfill \Box

Proposition 8.4.1 proves that the only attributes required by a satisfiable embedding are from the columns with null markers. Since we only consider
the concise forms of eFDs, an e-traversal for \( X \rightarrow Y \) over \( R \) only has to search through the attribute lattice over \( R(\bot) - X \). Proposition 8.4.2 gives the necessary and sufficient conditions when an eFD has satisfiable embeddings. In other words, if these conditions are not met, the entire e-traversal for \( X \rightarrow Y \) can be skipped since all possible embeddings will be violated.

Algorithm 21 LHS-traversal

1: **Input:** relation \( r \) over \( R \), the l-root \( l\_root \) of an eFD-tree
2: **Output:** eFDs satisfied by \( r \)
3: \( l \leftarrow 0 \)
4: candidates \( \leftarrow \{\emptyset\} \)
5: rhs[\emptyset] \( \leftarrow R \)
6: while |candidates| > 0 do
7:    for each \( X \in \text{candidates} \) do
8:       \( Y' \leftarrow \{A \in \text{rhs}[X] \mid r \models XA : X \rightarrow A\} \)
9:       \( Y'' \leftarrow \{A \in \text{rhs}[X] \mid r \models R : X \rightarrow A\} - Y' \)
10:      if |\( Y'' \)| > 0 then
11:         e_traversal\((r, R, l\_root, X, Y'', \pi_X)\)
12:         rhs[X] \( \leftarrow \text{rhs}[X] - Y' \)
13:     \( l \leftarrow l + 1 \)
14:     candidates \( \leftarrow \{X \subseteq R \mid |X| = l\} \)
15:     for each \( X \in \text{candidates} \) do
16:        \( Y' \leftarrow \bigcap\{\text{rhs}[X'] \mid X' \subseteq X \land \text{rhs}[X'] \neq \emptyset\} - X \)
17:        \( Y' \leftarrow \{A \in Y' \mid XA : X \rightarrow A \text{ is not redundant in the eFD-tree of } l\_root\} \)
18:        if |\( Y' \)| > 0 then
19:           rhs[X] \( \leftarrow Y' \)
20:        else
21:           Remove \( X \) from candidates
22: return \( l\_root \)

Algorithm 21 and 22 demonstrate how the row-efficient algorithm works in a conceptual level. The l-traversal algorithm starts at first to search through all possible LHSs. Meanwhile each LHS candidate is mapped to the set of all possible RHS attributes. As the level goes up in the l-traversal, sizes of LHS candidates become larger but the size of the RHS assigned to a LHS becomes smaller. For each pair of LHS and RHS candidate, an e-traversal is executed accordingly to discover all possible embeddings. E-traversals are similar to l-traversals but instead they only examine and validate embeddings with a fixed LHS. After e-traversals (step 11), the l-traversal will generate new pairs of LHS and RHS candidates for the next level. A new LHS candidate must be a superset of some LHS candidate at the current level (step 14). The new RHS candidate of a new LHS must be the intersection of all
Algorithm 22 E-traversal

1: **Input:** relation \( r \) over \( R \), the l-root \( l_{\text{root}} \) of an eFD-tree, LHS attributes \( X \), RHS attributes \( Y \), stripped partition \( \pi_X \)

2: **Output:** updates on the eFD-tree of \( l_{\text{root}} \) with all non-redundant \( E \subseteq R \) s.t. \( r \models E : X \rightarrow Y \)

3: \( l \leftarrow 0 \)

4: \( \text{candidates} \leftarrow \{ \varnothing \} \)

5: \( \text{rhs}[\varnothing] \leftarrow Y \)

6: **while** \( \text{candidates} > 0 \) **do**

7: **for each** \( E \in \text{candidates} \) **do**

8: \( Y' \leftarrow \{ A \in \text{rhs}[X] \mid r \models EXA : X \rightarrow A \} \)

9: **if** \( |Y'| > 0 \) **then**

10: Insert \( E - XY' : X \rightarrow Y' \) into \( l_{\text{root}} \)

11: \( \text{rhs}[X] \leftarrow \text{rhs}[X] - Y' \)

12: \( l \leftarrow l + 1 \)

13: \( \text{candidates} \leftarrow \{ E \subseteq R(\bot) - X \mid |E| = l \} \)

14: **for each** \( E \in \text{candidates} \) **do**

15: \( Y' \leftarrow \cap \{ E' \mid E' \subseteq E \wedge \text{rhs}[E'] \neq \varnothing \} - E \)

16: \( Y' \leftarrow \{ A \in Y' : E : X \rightarrow A \text{ is not redundant in the eFD-tree of } l_{\text{root}} \} \)

17: **if** \( |Y'| > 0 \) **then**

18: \( \text{rhs}[E] \leftarrow Y' \)

19: **else**

20: Remove \( E \) from \( \text{candidates} \)

21: **return** \( l_{\text{root}} \)

the RHSs whose corresponding LHSs are a subset of the new LHS (step 17).

For example, if a RHS attribute is not in the intersection, the attribute must belong to the RHS of some validated eFD at lower levels.

Before executing an e-traversal, we use two heuristics to help reduce the search space of eFDs. Firstly, we verify which attributes from the given RHS are satisfiable (step 8). If there is a non-empty and satisfiable subset of the given RHS, then it is no longer necessary to search for larger LHS candidates for the verified RHS (step 12). This is because any superset of the LHS with the verified RHS is satisfiable. Secondly, we verify which attributes from the given RHS meet the conditions in Proposition 8.4.2 (step 9). In this way, it makes sure at least one satisfiable eFD can be found during an e-traversal.

Note that eFDs are discovered in their concise forms. So, during the l-traversal, each RHS candidate should not intersect with the LHS where it is assigned. For example, if \( X = \{ A, B, C \} \) and \( Y' = \{ C, D \} \), then \( E : ABC \rightarrow C \) is trivial for any \( E \). Similarly, during an e-traversal, each RHS candidate should not intersect with the embedding where it is assigned. For example,
if $E = \{A, B, C\}$ and $Y' = \{C, D\}$, then it is equivalent to the situation where $E = \{A, B\}$ and $Y' = \{C, D\}$, which should have been examined at level 2 or even lower levels of the embedding attribute lattice.

Finally, we address important details on how to generate candidates of an attribute lattice ($LHS$s or embeddings) level-wisely and validate eFDs. To populate candidates in an attribute lattice level-wisely, it is more efficient to use prefix blocks [52]. Step 14 in Algorithm 21 and step 13 in Algorithm 22 only describes what candidates are at each level for the ease of understanding but they should be replaced with an implementation of prefix blocks. To validate eFDs, stripped partitions can be efficiently utilized which has been shown in Algorithm 14. Since LHS candidates are generated level-wisely, stripped partitions are computed incrementally.

Algorithm 23 Validation

1: Input: relation $r$ over $R$, eFD $E : X \rightarrow A$, stripped partition $\pi_X$
2: Output: true if $r \models E : X \rightarrow A$
3: for each $S \in \pi_X$ do
4:   $S^E \leftarrow \emptyset$
5:   for each $t \in S$ do
6:     if $\forall A \in E : t(A) \neq \bot$ then
7:       $S^E \leftarrow S^E \cup \{t\}$
8:   if $|S^E| > 0$ then
9:     $S^E \leftarrow \{t_1, \ldots, t_n\}$
10:    for each $t_i \in S^E$ where $i > 1$ do
11:       if $t_i(A) \neq t_1(A)$ then
12:         return false
13: return true

In summary, Algorithm 21 enumerates all possible eFDs by traversing attribute lattices of LHSs and embeddings so that these eFDs can be validated by the given relation. An l-traversal searches through all possible LHSs of different sizes from small to large. In such way, some eFD with a larger LHS can be pruned since they are redundant to some previous eFD with a smaller LHS. Given a LHS, an e-traversal searches through all possible embeddings for the fixed LHS. Eventually, every possible eFD will be either validated or violated or redundant. A validated eFD will be stored to help eliminate redundant eFDs discovered later; a violated eFD will enlarge the size of its LHS.
or embedding by 1; a redundant eFD will be discarded. Algorithm 21 terminates if either the rest of search space becomes redundant or the l-traversal has finished examining the maximum level of the LHSs’ attribute lattice. This concludes the correctness of Algorithm 21.

**Theorem 8.4.3 (Row-efficient EFD Discovery).** Algorithm 21 computes the canonical cover of all eFDs that are satisfied by the given relation.

### 8.5 Column-efficient Discovery

Column-efficient algorithms take a data-driven approach to the discovery problems. This class of algorithms extracts counter-examples from a given relation and uses them to derive true data dependencies. By following this direction, we introduce the column-efficient (row-based) discovery algorithm for eFDs.

First of all, we define the notion of non-eFDs which are used to represent violations of eFDs in a given relation.

**Definition 8.3 (Non-eFD)** A non-eFD over relation schema R is the expression $E : X \not\rightarrow Y$ where $X, Y \subseteq E$ and $X \cap Y = \emptyset$. We say a non-eFD $E : X \not\rightarrow Y$ is a violation or counter-example with respect to relation r over R if and only if there are tuples $t_1, t_2 \in r^E$ such that $t_1(X) = t_2(X)$ and $t_1(B) \neq t_2(B)$ for all $B \in Y$.

A non-eFD $E : X \not\rightarrow Y$ restricts the forms of eFDs in terms of embeddings, LHSs and RHSs. Firstly, for all $E' \subseteq E$ and $XY \subseteq E'$, eFD $E' : X \rightarrow Y$ cannot be satisfiable. Secondly, for all $X' \subseteq X$, eFD $E : X' \rightarrow Y$ cannot be satisfiable. Thirdly, for all $Y' \subseteq Y$, eFD $E : X \rightarrow Y'$ cannot be satisfiable. So, we say a non-eFD $E' : X' \not\rightarrow Y'$ contradicts an eFD $E : X \rightarrow Y$ if and only if $E \subseteq E'$, $X \subseteq X'$ and $Y \subseteq Y'$. Given a set of eFDs, if there is a non-eFD that contradicts some of the eFDs, then one can augment the contradicted eFDs to derive new
eFDs. Proposition 8.5.1 shows that true eFDs can be derived eventually if all the non-eFDs from the given relation no longer contradict any of the derived eFDs. In Example 8.2, we further demonstrate how to augment an eFD with respect to a non-eFD.

**Proposition 8.5.1 (Non-EFD).** Let $r$ be a relation over $R$ and $\Sigma^{-1}$ the set of all non-eFDs in $r$. An eFD $E : X \rightarrow Y$ is satisfied by $r$ if and only if there is no non-eFD $E' : X' \not\rightarrow Y' \in \Sigma^{-1}$ such that $E \subseteq E'$, $X \subseteq X'$, and $Y \subseteq Y'$.

**Proof.** Let $r$ be a relation over $R$ and $\Sigma^{-1}$ the set of all non-eFDs in $r$.

Case $\Rightarrow$: Suppose $r$ satisfies eFD $E : X \rightarrow Y$. Assume there is a non-eFD $E' : X' \not\rightarrow Y' \in \Sigma^{-1}$ where $E \subseteq E'$, $X \subseteq X'$ and $Y \subseteq Y'$. Then, there must be tuples $t_1, t_2 \in r^{E'}$ such that $t_1(X') = t_2(X')$ and $t_1(A) \neq t_2(A)$ for all $A \in Y'$. However, $t_1(X) = t_2(X)$ and $t_1(A) \neq t_2(A)$ for all $A \in Y$ since $t_1, t_2 \in r^{E} \subseteq r^{E'}$, $X \subseteq X'$ and $Y \subseteq Y'$, which contradicts $r \not\models E : X \rightarrow Y$.

Case $\Leftarrow$: Suppose $r$ does not satisfy eFD $E : X \rightarrow Y$. There must exist tuples $t_1, t_2 \in r^E$ such that $t_1(X) = t_2(X)$ and $t_1(A) \neq t_2(A)$ for all $A \in Y$.

\[\square\]

**Example 8.2 (Derivation of EFD)** Let $ABC : A \rightarrow BC$ be an eFD over relation schema $R = \{A, B, C, D, E\}$. Suppose $ABCD : A \not\rightarrow B$ is a non-eFD. By Definition 8.3, eFD $ABC : A \rightarrow B$ must be violated. To surpass the violation, firstly, we can augment the violated eFD by either enlarging its embedding or LHS. Adding attributes from $R - ABC$ to the violated embedding ($ABC$) creates a non-contradicting eFD while keeping the original LHS i.e. $ABCE, A \rightarrow B$; adding attributes from $ABCD - AB$ to the violated LHS ($A$) creates non-contradicting eFDs i.e. $ABC : AC \rightarrow B$ and $ABCD : AD \rightarrow B$.

\[\square\]

Computing all non-eFDs simply needs to check values between each pair of tuples in the given relation. Despite duplications, many non-eFDs are redundant to one another. For example, given a non-eFD $E : X \not\rightarrow Y$, one certainly should not have to consider non-eFD $E' : X' \not\rightarrow Y$ where $E' \subseteq E$.
and $X' \subseteq X$. Ignoring redundancies among non-eFDs is problematic for the column-efficient algorithm because using redundant non-eFDs results in more steps to derive true eFDs (see Example 8.3). Fortunately, redundant non-eFDs can be easily eliminated using an eFD-tree. Algorithm 24 demonstrates how non-redundant non-eFDs are computed given a relation. The algorithm firstly sorts all the unique non-eFDs extracted from a relation by their sizes of embeddings and LHSs (step 13). By inserting smaller non-eFDs firstly, it allows larger non-eFDs that are inserted later (step 16) to remove smaller but redundant ones by utilizing the fast search capabilities of the eFD-tree (step 15). The removal process can be easily adapted from Algorithm 20 so we do not give more details.

**Example 8.3 (Redundant Non-eFD)** Let $ABC : A \not\rightarrow B$ and $ABC : AC \not\rightarrow B$ be non-eFDs over relation schema $R = \{A, B, C, D\}$. Suppose an eFD $AB : A \rightarrow B$. Obviously, both non-eFDs conflict with the eFD. If non-eFD $ABC : A \not\rightarrow B$ is used to augment the eFD first, new eFD $ABC : AC \rightarrow B$ will be derived. However, the second non-eFD still contradicts the new eFD. In this case, the first eFD is redundant since the second non-eFD contains additional information on how to reshape the LHS of an eFD.

**Algorithm 24** non-redundant non-eFD

1. **Input:** relation $r$ over $R$
2. **Output:** the set of all non-redundant non-eFDs
3. Let relation $r$ be the form $\{t_1, t_2, \ldots, t_n\}$
4. $\Sigma^{-1} \leftarrow \emptyset$
5. Let $T$ be an empty eFD-tree
6. for each $t_i \in r$ do
7. $E \leftarrow \emptyset$, $X \leftarrow \emptyset$, $Y \leftarrow \emptyset$
8. for each $t_j \in r$ where $i+1 \leq j \leq n$ do
9. $E \leftarrow \{A \in R \mid t_i(A) \neq \bot \neq t_j(A)\}$
10. $X \leftarrow \{A \in E \mid t_i(A) = t_j(A)\}$
11. $Y \leftarrow E - X$
12. $\Sigma^{-1} = \Sigma^{-1} \cup \{(E, X \not\rightarrow Y)\}$
13. Sort $\Sigma^{-1}$ in an ascending order by the sizes of LHSs firstly and embeddings secondly
14. for each $(E, X \not\rightarrow Y) \in \Sigma^{-1}$ do
15. Remove all $(E', X' \not\rightarrow A) \in T$ for all $A \in Y$ where $E' \subseteq E$ and $X' \subseteq X$
16. Insert $E : X \not\rightarrow Y$ into $T$
17. return $\{E : X \not\rightarrow Y \in T\}$
Finally, we demonstrate the column-efficient algorithm in Algorithm 25. Note that all eFDs and non-eFDs described in the algorithms are in their concise forms since it is critical to the runtime performance. The algorithm firstly computes the set of non-eFDs from the given relation using Algorithm 24. To initialize the eFD-tree, only the most general eFD is inserted i.e. $\emptyset : \emptyset \rightarrow R$. Next, each non-eFD is applied iteratively to the eFD-tree so that non-contradicting eFDs can be derived. For each eFD contradicted by a non-eFD, it will be removed from the tree. Meanwhile, new eFDs are derived by augmenting the removed eFD with respect to the contradicting non-eFD. The newly derived eFDs are then inserted to the eFD-tree only if they are not redundant to the existing eFDs in the tree. Proposition 8.5.2 proves that augmenting eFDs based a new non-eFD always compute a new canonical cover of eFDs with respect to all the considered non-eFDs.

**Proposition 8.5.2** (Derivation of EFD). Let $\Gamma^{-1} = \Sigma^{-1} \cup \{E : X \not\rightarrow A\}$ be the set of all non-eFDs of relation $r$ over $R$. $\Gamma$ and $\Sigma$ are the canonical cover of eFDs with respect to $\Gamma^{-1}$ and $\Sigma^{-1}$. For all $E' : X' \rightarrow A' \in \Sigma$ the following must hold:

1. $E' : X' \rightarrow A' \in \Gamma$, or

2. $E'B : X' \rightarrow A'$ is in or redundant to $\Gamma$ for all $B \in R - E$, and

3. $E'B : X'B \rightarrow A'$ is in or redundant to $\Gamma$ for all $B \in E - X$, and

4. $E'B : X' \rightarrow A' \not\in \Gamma$ for all $B \in E$, and

5. $E'B : X'B \rightarrow A' \not\in \Gamma$ for all $B \in X$, and

6. $E'B : X'B \rightarrow A'$ is redundant to $\Gamma$ for all $B \in R - E$

**Proof.** Let $\Gamma^{-1} = \Sigma^{-1} \cup \{E : X \not\rightarrow A\}$ be the set of all non-eFDs of relation $r$ over $R$. $\Gamma$ and $\Sigma$ are the canonical cover of eFDs with respect to $\Gamma^{-1}$ and $\Sigma^{-1}$. Take any $E' : X' \rightarrow A' \in \Sigma$

If $A \neq A'$ or $E' \not\subseteq E$ or $X' \not\subseteq X$, then $E' : X' \rightarrow A' \in \Gamma$ because $E' : X' \rightarrow A'$ is already a canonical eFD in $\Sigma$ and it does not imply non-eFD $E : X \rightarrow A$. 


Suppose \( A = A' \) and \( E' \subseteq E \) and \( X' \subseteq X \). Take any \( B \in R - E \). Since \( E' : X' \rightarrow A' \in \Sigma \), there exists no \( E'' : X'' \not\rightarrow A' \in \Sigma^{-1} \) by Proposition 8.5.1 where \( E' \subseteq E'' \) and \( X' \subseteq X'' \). Knowing \( E' \subseteq E'B \), hence, \( E'B : X' \rightarrow A' \) is a satisfiable eFD with respect to \( \Sigma^{-1} \). Furthermore, \( E'B : X' \rightarrow A' \) is a satisfiable eFD with respect to \( \Gamma^{-1} \) because \( B \not\in E \) and \( E'B \not\subseteq E \). If there exists \( E'' : X'' \rightarrow A' \in \Sigma \) where \( E'' \subseteq E'B \) and \( X'' \subseteq X' \), then \( E'' \not\subseteq E \) because \( B \) must be in \( E'' \). If \( B \not\in E'' \) and \( E'' \subseteq E' \), then \( E'' : X'' \rightarrow A' \) and \( E' : X' \rightarrow A' \) are both in \( \Sigma \) and redundant to each other, which draws a contradiction. So, \( E'' \not\subseteq E \) and \( E'' : X'' \rightarrow A' \in \Gamma \). Namely, \( E'B : X' \rightarrow A' \) is redundant to \( \Gamma \). Otherwise, \( E'B : X' \rightarrow A' \in \Gamma \) if it is not implied by other eFD in \( \Sigma \). Hence, claim 2 is proven.

Take any \( B \in E - XA' \). Since \( E' : X' \rightarrow A' \in \Sigma \), there exists no \( E'' : X'' \not\rightarrow A' \in \Sigma^{-1} \) where \( E' \subseteq E'' \) and \( X' \subseteq X'' \) by Proposition 8.5.1. Knowing \( E' \subseteq E'B \) and \( X' \subseteq X'B \), hence, \( E'B : X'B \rightarrow A' \) is a satisfiable eFD with respect to \( \Sigma^{-1} \). Furthermore, \( E'B : X'B \rightarrow A' \) is a satisfiable eFD with respect to \( \Gamma^{-1} \) because \( B \not\in X \) and \( X'B \not\subseteq X \). Suppose there exists \( E'' : X'' \rightarrow A' \in \Sigma \) where \( E'' \subseteq E'B \) and \( X'' \subseteq X'B \).

Case \( B \in X'' \): \( X'' \not\subseteq X \) implies \( E'' : X'' \rightarrow A' \in \Gamma \). Therefore, \( E'B : X'B \rightarrow A' \) becomes redundant to \( \Gamma \).

Case \( B \not\in X'' \): we can conclude that \( E'' \subseteq E'B \subseteq E \) and \( X'' \subseteq X' \subseteq X, X'B \). Since \( E'' \subseteq E \) and \( X'' \subseteq X \), \( E'' : X'' \rightarrow A' \) is not in \( \Gamma \) but \( E''B : X''B \rightarrow A' \) is satisfiable with respect to \( \Gamma^{-1} \). Since \( \Sigma \) is finite, we can always choose \( E'' \) and \( X'' \) where there exists no \( E''' \subseteq E''B \) or \( X''' \subseteq X''B \) such that \( E''' : X''' \rightarrow A' \) where \( A' \in \Sigma \). To choose such \( E'' : X'' \rightarrow A \), we can keep replacing \( E''' : X''' \rightarrow A' \) with \( E'' : X'' \rightarrow A' \) if \( E''' : X''' \rightarrow A' \) where \( E''' \subseteq E''B \) and \( X''' \subseteq X''B \) are found in \( \Sigma \) because \( E''' \subseteq E''B \subseteq E'B \) and \( X''' \subseteq X''B \subseteq X'B \). By choosing such \( E'' \) and \( X'' \) with respect to \( B \), it is guaranteed that \( E''B, X''B \rightarrow A \in \Gamma \), which further concludes that \( E'B : X'B \rightarrow A' \) is redundant to \( \Gamma \). Otherwise, \( E'B : X'B \rightarrow A' \in \Gamma \) if it is not implied by other eFDs in \( \Sigma \). Hence claim 3 is proven.
8.5. Column-efficient Discovery

Take any $B \in E$. Since $E'B \subseteq E, E'B : X' \rightarrow A'$ is not a satisfiable eFD with respect to $\Gamma^{-1}$ so it is not in $\Gamma$. Hence, claim 4 is proven.

Take any $B \in X$. Since $E'B \subseteq E$ and $X'B \subseteq X, E'B : X'B \rightarrow A'$ is not a satisfiable eFD with respect to $\Gamma^{-1}$ so it is not in $\Gamma$. Hence, claim 5 is proven.

Take any $B \in R - E$. If $E'B : X'B \rightarrow A'$ is not implied by any other eFDs in $\Sigma$, it is still implied by $E'B : X' \rightarrow A'$ as $B \notin E'$. In other words, $E'B : X'B \rightarrow A' \notin \Gamma$ for all $B \in R - E$. Hence, claim 6 is proven.

\[ \square \]

\textbf{Algorithm 25} Column-efficient algorithm

1. \textbf{Input:} relation $r$ over $R$
2. \textbf{Output:} the canonical cover of all eFDs satisfied by $r$
3. Let $\Sigma^{-1}$ be the set of all non-redundant non-eFDs
4. Let $T$ be an empty eFD-tree
5. Insert eFD $\emptyset : \emptyset \rightarrow R$ into $T$
6. Sort $\Sigma^{-1}$ in ascending order by the sizes of LHSs firstly and embeddings secondly
7. for each $E : X \rightarrow Y \in \Sigma^{-1}$ do
8. for each $E' : X' \rightarrow Y' \in T$ where $E'X' \subseteq EX, X \subseteq X'$, and $Y' \subseteq Y$ do
9. Remove $E' : X' \rightarrow Y'$ from $T$
10. for each $A \in R - EXY$ do
11. if $E'A : X' \rightarrow Y'$ is not redundant in $T$ then
12. Insert $E'A : X' \rightarrow Y'$ into $T$
13. for each $A \in EY$ do
14. $E'' \leftarrow E', Y'' \leftarrow Y'$
15. if $A \notin Y'$ then
16. $E'' \leftarrow E''A$
17. else
18. $Y'' \leftarrow Y'' - \{A\}$
19. if $Y'' = \emptyset$ then continue
20. if $E'' : X'A \rightarrow Y''$ is not redundant in $T$ then
21. Insert $E'' : X'A \rightarrow Y''$ into $T$
22. return $\{(E, X \rightarrow Y) \in T\}$

Algorithm 25 builds up true eFDs from the most general eFD by augmenting embeddings and LHSs of false eFDs iteratively. The correctness can be guaranteed because all the augmentations eventually avoid the newly derived eFDs being contradicted by any given non-eFDs. Besides, Proposition 8.5.2 can be used to show that the algorithm eventually computes the canonical cover of eFDs with respect to all the considered non-eFD. Hence, this concludes the correctness of Algorithm 25.
Theorem 8.5.3 (Column-efficient EFD Discovery). Algorithm 25 computes the canonical cover of all eFDs that are satisfied by the given relation.

8.6 Hybrid Discovery

The row-efficient and column-efficient algorithm are only efficient for datasets with either a large number of rows or a large number of columns. However, the number of columns and rows sometimes both can be large in most of the real-world datasets. In fact, row-efficient and column-efficient algorithms can be hybridized to discover data dependencies as well to overcome limits caused by large numbers of rows and columns. In this section, we compose the aforementioned row-efficient and column-efficient eFD discovery algorithms and introduce the hybrid discovery algorithm for eFD discoveries.

The row-efficient and the column-efficient algorithm both suffer from severe drawbacks while dealing with large real-world datasets. For instance, given a real-world dataset, the row-efficient algorithm not only has to search
8.6. Hybrid Discovery

through a very large attribute lattice but also requires more memory to store stripped partitions during the search; the column-efficient algorithm needs to process a large collection of duplicated and redundant non-eFDs whose size is quadratic to the number of rows of the given relation. Fortunately, merits of the two algorithms compensate each other’s drawbacks. On one hand, the row-efficient algorithm can use non-eFDs to populate candidates in the next level instead of an exhaustive enumeration. On the other hand, the column-efficient algorithm can directly validate an eFD with the corresponding stripped partition without checking through all the non-eFDs.

Figure 8.5 illustrates how our proposed hybrid algorithm effectively orchestrates the row-efficient and column-efficient algorithm. The hybrid algorithm takes a relation \( r \) over schema \( R \) as an input. Then, it initializes an eFD-tree and a stripped partition based \( R \) and \( r \). At the beginning of an iteration, all of the LHS and RHS candidates at level \( l \) of the eFD-tree must be retrieved. Meanwhile, a set of stripped partitions for the LHS candidates is computed using the stripped partitions from the previous iteration. Next, hybrid e-discoveries are executed according to each pair of LHS and RHS candidate. After a hybrid e-discovery is finished, it discovers all the embeddings for a given LHS, updates LHSs at level \( l + 1 \), and saves all the used non-eFDs. After the non-eFDs are collected from all the hybrid e-discoveries, they will be further refined by a redundancy cleaner (similar to Algorithm 24) So, at the end of an iteration, efficient updates can be made to the eFD-tree globally with the cleaned non-eFDs. Eventually, the hybrid algorithm terminates.
when it has iterated through all levels of the eFD-tree and no more updates can be made to the eFD-tree.

**Hybrid e-discovery.** As shown in Figure 8.5, embeddings of a given pair of LHS and RHS candidates are computed by a separate process called *hybrid e-discovery*. The process discovers the embeddings of a given LHS in a hybrid manner as well. The algorithm iteratively validates embedding candidates while updating its local search space (an attribute lattice of embeddings) as shown in Figure 8.6. The search space of a hybrid e-discovery is just a subtree that starts from an e-root. The e-root can be easily retrieved since the LHS candidate in the global eFD-tree is given. At the beginning of an iteration, embedding candidates will be retrieved and validated using the given stripped partition. If any violations are detected, the validation process will output the conflicting non-eFDs to the subsequent processes. After validations, the hybrid e-discovery needs to decide whether more non-eFDs should be sampled from the stripped partition to prune the current search space. Such decision depends on the percentage of invalidated eFDs over all the examined candidates. If there are too many invalidations e.g. over 1%, the hybrid e-discovery will run a *completeness-driven* non-eFD sampler. At the end of each iteration, all of the non-eFDs extracted from the validation and sampler will be used by the local induction process to update the local eFD-tree. The induction process in a hybrid e-discovery is *local*. It only uses non-eFDs to update embeddings of the given LHS and possibly generate new eFDs whose LHSs are only one attribute more than the given LHS candidate.

Hybrid e-discoveries implement two essential heuristics that make the hybrid algorithm much more practical on the large datasets (see Section 8.7). Firstly, non-eFDs are extracted with a *completeness-driven sampling* method. The driving idea is that the generality in the embedding of a non-eFD is more important when it comes to update eFDs with a fixed LHS i.e. as shown in Example 8.2, non-eFDs with larger embeddings and LHSs are more efficient
to derive true eFDs. Since LHS candidate is fixed and the stripped partition of the LHS is provided, one can easily find efficient non-eFDs to update the given embeddings by examining the pairs of tuples with less missing values. Algorithm 26 demonstrates a single run of the completeness-driven sampling. Note that, as mentioned in Figure 8.5, all tuples in $\pi_\varnothing(r)$ are sorted in advance by their completeness i.e. tuples with less missing values will be sampled first. Since stripped partitions are generated in an incremental manner, tuples in the stripped partitions that are generated later will remain the same order with respect to their completenesses. Therefore, Algorithm 26 always finds non-eFDs with larger embeddings first. In a hybrid e-discovery, if the non-eFD sampler is invoked, it keeps running Algorithm 26 until the number of newly sampled non-eFDs is below certain threshold e.g. the initial threshold is 1.0. Such threshold represents a point when there are enough non-eFDs to update the current search space effectively. If the non-eFD sampler is invoked again in the same hybrid e-discovery, which means the previous sampling is not sufficient, then the threshold will be decreased by half before the non-eFD sampler is executed. In this way, more new non-eFDs can be used for pruning when there are too many invalidated candidates that try to explode the search space. Secondly, there is no absolutely good way to find non-eFDs that can efficiently update LHSs of eFD candidates because LHSs of eFDs are normally domain dependent. In order to improve the efficiency of updating LHSs, a hybrid e-discovery will not update eFDs globally with extracted non-eFDs. Instead, all hybrid e-discoveries in the same iteration will be coordinated by a redundancy cleaner (as shown in Figure 8.5), which removes all the redundant non-eFDs. Overall, the first heuristic dramatically reduces the number of steps to derive true eFDs and the second heuristic saves the memory cost by preventing excessive updates on the global eFD-tree.

Algorithm 21 eventually computes the canonical cover of eFDs from a
Algorithm 26 Completeness-driven Sampling

1: **Input:** a stripped partition \( \pi_X(r) \), offset \( i \), relation \( r \) over \( R \)
2: **Output:** non-eFDs
3: \( \Sigma^{-1} = \emptyset \)
4: for each \( S \in \pi_X(r) \) do
5:   Let \( S = \{t_1, \ldots, t_n\} \)
6:   if \( i > 1 \) and \( t_i(A) \neq \bot \) for \( A \in R \) then
7:     continue
8:   for each \( j > i \) and \( j \leq n \) do
9:     \( E \leftarrow \{A \in R : t_i(A) \neq \bot \neq t_j(A)\} \)
10:    \( X' \leftarrow \{A \in E : t_i(A) = t_j(A)\} \)
11:    \( Y \leftarrow \{A \in E : t_i(A) \neq t_j(A)\} \)
12:    \( \Sigma^{-1} \leftarrow \Sigma^{-1} \cup \{(E', X \notightarrow Y)\} \)
13: return \( \Sigma^{-1} \)

given relation. The hybrid algorithm performs a breath-first search to traverse all eFDs. During the traversals, eFDs are validated along the way with the method implemented in the row-efficient algorithm. When invalid eFDs are identified, the eFD-tree will be updated with respect to a set of new non-eFDs. The algorithms that validate and update eFDs have been proven correct in Section 8.4 and 8.5 respectively. Hence, the correctness of the hybrid algorithm is concluded.

8.7 Experiments

In our experiments, we show the performance of the proposed algorithms discovering eFDs from the real-world benchmark datasets. Particularly, we show in-depth analysis to the algorithms’ runtime, memory consumption, and scalability. We have implemented all the algorithms in Visual C++ and conduct all the experiments on an Intel Xeon 3.6 GHz, 256 GB RAM, Windows 10 Dell workstation. The benchmark datasets are from the UCI machine learning data repository \(^1\) and previous research [87].

8.7.1 Performance on Real-world Data

Table 8.1 summarizes our results of eFD discoveries on the benchmark data. For each dataset, we show its number of rows (\#R), columns (\#C), incomplete

\(^1\)https://archive.ics.uci.edu/ml/
8.7. Experiments

<table>
<thead>
<tr>
<th>Datasets</th>
<th>#IR</th>
<th>#C</th>
<th>#IR</th>
<th>#C</th>
<th>#I⊥</th>
<th>#FDs</th>
<th>#non-eFDs</th>
<th>#eFDs</th>
<th>Row-efficient</th>
<th>Column-efficient</th>
<th>Hybrid</th>
<th>FD</th>
</tr>
</thead>
<tbody>
<tr>
<td>bridges</td>
<td>108</td>
<td>13</td>
<td>36</td>
<td>9</td>
<td>77</td>
<td>142</td>
<td>73</td>
<td>169</td>
<td>0.021</td>
<td>0.014</td>
<td>0.035</td>
<td>0.1</td>
</tr>
<tr>
<td>echo</td>
<td>132</td>
<td>13</td>
<td>71</td>
<td>12</td>
<td>132</td>
<td>527</td>
<td>89</td>
<td>438</td>
<td>0.034</td>
<td>0.013</td>
<td>0.026</td>
<td>0.1</td>
</tr>
<tr>
<td>hepatitis</td>
<td>155</td>
<td>20</td>
<td>75</td>
<td>15</td>
<td>167</td>
<td>8250</td>
<td>696</td>
<td>11887</td>
<td>322.107</td>
<td>0.287</td>
<td>0.431</td>
<td>0.6</td>
</tr>
<tr>
<td>horse</td>
<td>300</td>
<td>28</td>
<td>294</td>
<td>21</td>
<td>169</td>
<td>128727</td>
<td>5975</td>
<td>1182385</td>
<td>&gt;4H</td>
<td>56.371</td>
<td>72.704</td>
<td>7.1</td>
</tr>
<tr>
<td>cancer</td>
<td>691</td>
<td>11</td>
<td>16</td>
<td>16</td>
<td>46</td>
<td>44</td>
<td>48</td>
<td>0.029</td>
<td>0.167</td>
<td>0.044</td>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td>plista</td>
<td>996</td>
<td>63</td>
<td>996</td>
<td>34</td>
<td>23317</td>
<td>178152</td>
<td>5876</td>
<td>298806</td>
<td>&gt;4H</td>
<td>27.569</td>
<td>55.818</td>
<td>21.8</td>
</tr>
<tr>
<td>ncvoter1k</td>
<td>1000</td>
<td>19</td>
<td>1000</td>
<td>5</td>
<td>2863</td>
<td>758</td>
<td>226</td>
<td>1443</td>
<td>0.132</td>
<td>0.514</td>
<td>0.082</td>
<td>0.4</td>
</tr>
<tr>
<td>flight</td>
<td>1000</td>
<td>109</td>
<td>1000</td>
<td>69</td>
<td>51938</td>
<td>982631</td>
<td>3202</td>
<td>2997244</td>
<td>&gt;4H</td>
<td>9.698</td>
<td>40.88</td>
<td>53.4</td>
</tr>
<tr>
<td>diabetic</td>
<td>101766</td>
<td>30</td>
<td>100732</td>
<td>7</td>
<td>192849</td>
<td>40195</td>
<td>35158</td>
<td>407767</td>
<td>&gt;4H</td>
<td>&gt;4H</td>
<td>7996.09</td>
<td>N/a</td>
</tr>
<tr>
<td>weather</td>
<td>262920</td>
<td>18</td>
<td>157985</td>
<td>12</td>
<td>416560</td>
<td>918</td>
<td>1819</td>
<td>8323</td>
<td>751.277</td>
<td>&gt;4H</td>
<td>131.247</td>
<td>N/a</td>
</tr>
<tr>
<td>uniprot512k</td>
<td>512000</td>
<td>30</td>
<td>192849</td>
<td>19</td>
<td>3799296</td>
<td>3703</td>
<td>468</td>
<td>194742</td>
<td>&gt;4H</td>
<td>&gt;4H</td>
<td>727.108</td>
<td>N/a</td>
</tr>
</tbody>
</table>

Table 8.1: Runtime (in seconds) of discovering eFDs from benchmark real-world datasets with different algorithms

rows (#IR), incomplete columns (#IC), null markers (#⊥), FDs (#FDs), non-redundant non-eFDs (#non-eFDs), eFDs in the canonical cover (#eFDs), and the time cost by different algorithms. For the datasets with less than 1,000 rows, the column-efficient algorithm always performs better. Particularly, the column-efficient algorithm yields a much better performance when the number of columns in a dataset is large e.g. plista, flight. Although the row-efficient algorithm does not perform better than others in any case, it still can compute a large dataset with over 260,000 rows (weather) much faster than the column-efficient algorithm. The hybrid algorithm does not always outperform the other two since it is essentially designed to handle large datasets such as diabetic, weather, uniprot512k. As we expected, the performances of the hybrid algorithm are satisfying e.g. it only takes less than 15 minutes to process the uniprot512k dataset, which contains a half million rows and 30 columns. We show the memory (MB) consumed by the algorithms when they were running experiments in Table 8.1. The row-efficient algorithm in fact is not memory-friendly since it uses many stripped partitions for eFD validations. On the other hand, the hybrid algorithm takes advantage of the heuristics implemented in hybrid e-discoveries so that it is quite memory-friendly even though stripped partitions are used. For example, the hybrid algorithm’s memory consumptions are less or only slightly more than the column-efficient algorithm’s.
To further characterize the practicality of the three algorithms, we show how each algorithm performs on different row-fragment or column-fragment of a dataset. Figure 8.7 shows how each algorithm’s runtime scales when a given dataset is increased by its number of rows or columns. Weather and ncvoter dataset both start with 1,000 rows. The hybrid algorithm and the row-efficient algorithm have comparable performance in both experiments up to 60,000 rows. The column-efficient algorithm is dramatically outperformed by the other two after 10,000 rows. To demonstrate column-efficiencies, we use a diabetic dataset with only 10,000 rows but the number of columns ranges from 5 to 45. The row-efficient algorithm performs well only up to 20 columns. The hybrid algorithm starts gaining more advantages than the column-efficient algorithm after when datasets are over 35 columns. More interestingly, the time scale of the hybrid algorithm has very similar pattern to the scale of eFDs discovered in the datasets but less steep. In addition, we show the memory scalability of the algorithms in Figure 8.8 with the same collection of datasets. The hybrid algorithm is more memory-friendly. On one hand, the hybrid algorithm reduces much more search space and uses less stripped partitions than the row-efficient algorithm. On the other hand,
the hybrid algorithm efficiently samples non-eFDs from a dataset but the
column-efficient algorithm has to compute all the non-eFDs from a dataset.
Table 8.1 shows that the number of non-eFDs in a dataset often exceeds the
number of rows quadratically, which will increase the memory consumed by
the column-efficient algorithm.

8.8 Armstrong Samples

While discovering data dependencies from a given relation, only some of the
tuples essentially determine what the true data dependencies are. In fact, a
much smaller but “perfect” data sample can always be computed given any
relation. Such data sample is “perfect” because it satisfies the same set of data
dependencies as the original relation does. Using smaller samples to repre-
sent data dependencies comes with many benefits. For example, domain
experts can easily query or directly interact with the data samples regarding
the discovered data dependencies. By quickly understanding the data and
data dependencies, domain experts are able to exchange feedbacks with data
stewards more frequently. In fact, the “perfect” data samples have been stud-
ied in the field of informative Armstrong databases a.k.a Armstrong samples
[34]. In Chapter 5, we have developed a useful tool for Armstrong samples
of eUCs to help with data cleansing process. To further advance the stud-
ies of Armstrong samples, we next demonstrate how to compute Armstrong
samples of eFDs from a given relation.

We first define Armstrong samples of eFDs.

**Definition 8.4 (Armstrong Sample of EFD)** Let \( r \) be a relation and \( \Sigma \) the set of
all eFDs of \( r \). An Armstrong sample with respect to \( \Sigma \) is a subset \( r' \subseteq r \) such that
\( r' \) only satisfies \( \Sigma \).
Chapter 8. Discovering Embedded Functional Dependencies

Figure 8.7: Time scalability of eFD discovery algorithms
Figure 8.8: Memory scalability of eFD discovery algorithms
Given a set of non-eFDs $\Sigma^{-1}$ of $\Sigma$, the Armstrong sample of $\Sigma$ over $r$ can be easily computed as the set

$$\bigcup_{E : X \not\rightarrow Y \in \Sigma^{-1}} \{ t_1, t_2 \in r^E | t_1(X) = t_2(X) \text{ and } \forall A \in Y : t_1(A) \neq t_2(A) \}.$$ 

As we have discussed before, non-eFDs may happen to be redundant to one another so the following discussions only focus on non-redundant ones.

Next, we illustrate two solutions to computing non-eFDs from a given relation. When the given relation has a small number of rows, the direct algorithm of computing all the non-eFDs has been shown in Algorithm 24. The algorithm simply computes all the non-eFDs by inspecting every pair of tuples and then only keeps non-redundant ones via an eFD-tree. Although this algorithm is only in time $O(n^2)$ where $n$ is the number of rows in the relation, real-world datasets can easily expand on rows e.g. $ncvoter$ dataset contains more than one million rows. In this case, the direct algorithm is no longer efficient. On the other hand, there is an indirect algorithm which can derive non-eFDs from the discovered eFDs as shown in Algorithm 27. Basically, the indirect algorithm holds a set of non-eFD candidates and keeps updating the non-eFDs iteratively against a given set of eFDs. The correctness of Algorithm 27 can be shown with Lemma 8.8.1. The indirect approach is mainly adopted from Algorithm 9, which computes Armstrong relations of eUCs.

**Lemma 8.8.1 (Derivation of Non-EFD).** Let $\Gamma = \Sigma \cup \{ E : X \rightarrow A \}$ be a set of eFDs over a relation schema $R$. If $E' : X' \not\rightarrow A' \in \Gamma^{-1}$, then the following claims must hold:

1. $E' : X' \not\rightarrow A' \in \Sigma^{-1}$, or

2. there exists $B \in X$ such that $E' : X'B \not\rightarrow A' \in \Sigma^{-1}$, or

3. there exists $B \in E$ such that $E'B : X' \not\rightarrow A' \in \Sigma^{-1}$, or

4. there exists $B \in E$ such that $E'B : X'B \not\rightarrow A' \in \Sigma^{-1}$. 

Proof. Let $\Gamma = \Sigma \cup \{E : X \rightarrow A\}$ be a set of eFDs over a relation schema $R$. Take any $E' : X' \not\rightarrow A' \in \Gamma^{-1}$.

Case $A \neq A'$: Assume $E' : X' \not\rightarrow A' \notin \Sigma^{-1}$. Since $E' : X' \not\rightarrow A' \in \Gamma^{-1}$, $E' : X' \not\rightarrow A'$ is a non-eFD of $\Sigma$ but redundant to some non-eFD $E'' : X'' \not\rightarrow A' \in \Sigma^{-1}$ where $E' \subseteq E''$ and $X' \subseteq X''$. However, $E'' : X'' \not\rightarrow A' \in \Gamma^{-1}$ since $A' \neq A$, which draws a contradiction to the fact $E' : X' \not\rightarrow A' \in \Gamma^{-1}$.

Case $E \not\subseteq E'$ and $X \not\subseteq X'$: Suppose $A = A'$. Assume $E' : X' \not\rightarrow A' \notin \Sigma^{-1}$. Since $E' : X' \not\rightarrow A' \in \Gamma^{-1}$, there must exist $E'' : X'' \not\rightarrow A' \in \Sigma^{-1}$ where $E' \subseteq E''$ and $X' \subseteq X''$ because of the assumption. $E'' : X'' \not\rightarrow A'$ must also be implied by $E : X \rightarrow A$ in order to get $E' : X' \rightarrow A' \in \Gamma^{-1}$. Taking any $B \in E - X$ from $E''$ or $B \in X$ from $X''$ would result in $E \subseteq E'$ or $X \subseteq X'$. Taking any $B \in X$ from $E''$ would not produce any new non-eFDs to $\Gamma^{-1}$ since they are redundant to the case where any $B \in X$ is taken from $X''$. Therefore, if $E \not\subseteq E'$ and $X \not\subseteq X'$, $E' : X' \not\rightarrow A'$ must be in $\Sigma^{-1}$.

Case $E \subseteq E'$: Suppose $A = A'$. Since $E \subseteq E'$ and $E' : X' \not\rightarrow A' \in \Gamma^{-1}$, then $X \not\subseteq X'$. If $E' : X'B \not\rightarrow A' \notin \Sigma^{-1}$ for all $B \in X$, then $E' : X' \not\rightarrow A' \in \Sigma^{-1}$ (claim 1) otherwise $E' : X' \rightarrow A'$ will not be in $\Gamma^{-1}$. If $E' : X'B \not\rightarrow A' \in \Sigma^{-1}$ for all $B \in X$, then claim 2 is proven.
Case $X \subseteq X'$: Suppose $A = A'$. Since $X \subseteq X'$ and $E' : X' \not\rightarrow A' \in \Gamma^{-1}$, then $E' \not\subseteq E$. Suppose $E' : X' \not\rightarrow A' \not\in \Sigma^{-1}$. Then, there exists $E'' : X'' \not\rightarrow A' \in \Sigma^{-1}$ where $E' \subseteq E''$ and $X' \subseteq X''$. Also, $E \subseteq E''$ and $X \subseteq X''$ should hold otherwise $E'' : X'' \not\rightarrow A' \in \Gamma^{-1}$ instead of $E' : X' \not\rightarrow A'$. Since $E \not\subseteq E'$ and $E \subseteq E''$, there must exists $B \in E$ such that $E'' - B = E'$. In other words, $E'B, X'' \not\rightarrow A' \in \Sigma^{-1}$. Furthermore, if $B \in E \cap X''$, then $X' = X'' - B$ otherwise $X' = X''$. Hence, claim 3 and 4 are proven.

Table 8.3 summarizes our results on computing eFD Armstrong samples for the real-world benchmark datasets. The table shows the relative sizes of the Armstrong samples and the times to computing them by two different algorithms. Obviously, the direct algorithm is faster when a dataset only has a small number of rows. On the other hand, the indirect algorithm only takes at most 10 minutes to compute Armstrong samples for the large datasets. It is also expected to see that the relative sample sizes are large when the original datasets are small e.g. horse dataset only contain 300 rows in total. As for large datasets such as uniprot, there are only 2,752 rows in the sample but a half million rows in the original. At last, we show how the relative sizes of Armstrong samples change over datasets with different numbers of rows. As shown in Figure 8.9, the relative sizes of Armstrong samples in both datasets have a logarithmic decrease while the number of rows are increased.

### 8.9 Summary

Embedded functional dependencies can play an important role in lifting classical applications of the relational model of data to modern applications of big data with missing information. Solutions to the discovery problem of eFDs will be integral for advancing the design of database schemata and data warehouses, data cleansing, and data profiling. In this chapter, we introduced the first row-efficient, column-efficient and hybrid algorithms that
8.9. Summary

<table>
<thead>
<tr>
<th>Dataset</th>
<th>#R</th>
<th>#⊥</th>
<th>%Sample Size</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Indirect</td>
<td>Direct</td>
</tr>
<tr>
<td>bridges</td>
<td>21</td>
<td>12</td>
<td>19%</td>
<td>0.007</td>
</tr>
<tr>
<td>echo</td>
<td>46</td>
<td>28</td>
<td>35%</td>
<td>0.007</td>
</tr>
<tr>
<td>hepatitis</td>
<td>96</td>
<td>44</td>
<td>62%</td>
<td>0.795</td>
</tr>
<tr>
<td>horse</td>
<td>219</td>
<td>902</td>
<td>73%</td>
<td>2366</td>
</tr>
<tr>
<td>breast</td>
<td>23</td>
<td>0</td>
<td>3%</td>
<td>0.002</td>
</tr>
<tr>
<td>plista</td>
<td>164</td>
<td>3969</td>
<td>16%</td>
<td>197</td>
</tr>
<tr>
<td>ncvoter</td>
<td>197</td>
<td>449</td>
<td>20%</td>
<td>0.04</td>
</tr>
<tr>
<td>flight</td>
<td>754</td>
<td>38816</td>
<td>75%</td>
<td>3548</td>
</tr>
<tr>
<td>diabetic</td>
<td>8009</td>
<td>10546</td>
<td>8%</td>
<td>456</td>
</tr>
<tr>
<td>weather</td>
<td>1517</td>
<td>318</td>
<td>0.58%</td>
<td>24</td>
</tr>
<tr>
<td>uniprot512k</td>
<td>2752</td>
<td>16491</td>
<td>0.54%</td>
<td>86.029</td>
</tr>
</tbody>
</table>

TABLE 8.3: Time (seconds) of computing eFD Armstrong samples

![Figure 8.9: Scalability of Armstrong sample sizes.](image-url)
solve the eFD discovery problem. The proposed algorithms work together as a toolkit in facilitating eFD discovery in practice. The row-efficient algorithm and column-efficient algorithm perform best on datasets they have been designed for, namely when the number of rows or columns in a dataset is small, respectively. They are recommended when data stewards or domain experts want to inspect eFDs of datasets that meet these requirements, such as data samples. Since their computation is fast on such datasets, users can effectively use them as part of architectures that address complex data management tasks. For large datasets, our hybrid algorithm typically offers low runtime and memory costs. The characteristics of these three algorithms have been demonstrated by our extensive experiments on real-world benchmark datasets. In particular, we compared runtime performance and memory consumption of the three algorithms on the benchmarks. We also tested their column- and row-scalability in terms of running time and memory consumption by discovering eFD from incremental fragments of a given dataset. In addition, we introduced Armstrong samples for eFDs. They can provide a useful aid to consolidate important domain knowledge within data models.
Chapter 9

Conclusion

Our overarching research question asked which notions of unique constraints and functional dependencies would be robust under different interpretations of missing values, allow users to stipulate data completeness requirements, and permit efficient solutions to important computational problems in data profiling and relational schema design. In this final chapter, we will briefly outline how the results in this thesis have addressed this research question, and what potential impact our contributions will have. We also list some possible avenues for future work.

9.1 Conclusion

The correct handling of incomplete data, specifically of missing data, has been both a fundamental and challenging problem to address for decades. It is fundamental as missing data occurs naturally and frequently, and even more so in modern day applications such as data integration and big data. The problem is challenging as it has resisted robust solutions since it has first surfaced in the 1980s. The problem occurs in many facets, including query languages and schema design. In particular, fundamentally important classes of integrity constraints need to be extended from the relational model of data to accommodate missing values. Such classes include keys and functional dependencies which address entity and also referential integrity of data. Previous work on such extensions have focused on popular
interpretations of missing values, such as “unknown at present” [46], “inapplicable” [30], and “no information” [112]. In terms of efficiency, a mix of such extensions often causes computational overheads that cannot be justified in practice. Hence, it is often a requirement to apply a uniform interpretation to all occurrences of missing values. The best example is the industry standard SQL, which handles all missing values via a uniform representation in the form of null marker occurrences. Such uniform treatment then causes serious conceptual problems because it is difficult to justify that every null marker occurrence can be interpreted in the same way. Again, such justification is even more difficult in modern application contexts such as data integration where incomplete data originates from different sources and null markers are introduced to meet the rigid structural constraints of the underlying data model. And even in the case where such an interpretation-specific approach can be justified, other important computational problems resist efficient solutions. These obstacles have clearly motivated our overarching research question.

As an answer we have proposed the novel class of embedded data dependencies, and exemplified its idea on unique constraints and functional dependencies. In particular, we focused on extending keys and functional dependencies to embedded unique constraints (eUCs) and embedded functional dependencies (eFDs), respectively. The first novelty of these notions arises from their semantics, which is robust under different interpretations of null marker occurrences. Indeed, the validity of the constraints within a given incomplete dataset is determined and determined only by the complete fragments of the given dataset. The second novelty of eUCs and eFDs is their capability of enabling users to declare both completeness and integrity requirements of applications, and to do that by a clear separation of concerns. Such capability promotes different data quality dimensions to first-class citizens of applications, for example in schema design or data profiling. Thirdly, the novelty
of these notions allows users to identify and prohibit rich sources of data redundancy. In fact, an eFD $E : X \rightarrow Y$ can cause redundant data values if and only if $E : X$ is not an eUC. Fourthly, eUCs and eFDs subsume well-known previous notions of keys and functional dependencies as special cases, including SQL unique constraints and classical functional dependencies. The thesis has illustrated the efficient use of eUCs and eFDs for many significant computational problems, including the implication problem, the computation of Armstrong samples, the discovery problem, schema design, integrity enforcement, data cleaning, business rule acquisition, and query optimization. The illustration includes a variety of techniques, ranging from theoretical foundations by the use of data structures, algorithm design, computational complexity, theorems and mathematical proofs, to practical considerations by the use of implementations of our algorithms, their experimental evaluation with synthetic and real-world benchmark schemata and data, the development of prototype systems including graphical user interfaces, as well the implementation within actual SQL-based relational database management systems.

Technically, Chapter 5 has established a sound and complete axiomatization of eUCs, a linear-time algorithm for deciding the implication of eUCs, which has been shown to be useful in data profiling and schema design applications. In addition, we have established structural and algorithmic characterizations of Armstrong relations for eUCs. Similar results were established for eFDs in Chapter 7 and Chapter 8. Solving the implication problems of eUCs and eFDs provides a strong reasoning tool for database schema design. With the help of Armstrong relations, database designers can easily communicate with domain experts to integrate reliable domain knowledge into their solutions, and acquire more integrity rules that are meaningful to their applications. Many popular data-driven applications such as data cleaning, query optimization, and data repository design require the discovery of meta-data profiles from existing data. In a modern day context,
these meta-data profiles need to be built upon huge volumes of incomplete data. Discovering eUCs and eFDs brings forward many new insights into data profiling. Firstly, the discovered eUCs and eFDs can provide a structured view of data (as exemplified in Section 8.1 of Chapter 8). Secondly, eFDs capture two data quality dimensions, namely data integrity and data completeness, when they are discovered from incomplete data. Thirdly, they can be used to build semantic profiles of data, which greatly help with the cleaning of dirty data and the acquisition of meaningful business rules (as demonstrated in Section 6.10 of Chapter 6). Lastly, also in Chapter 6, Section 6.8 shows a strong motivation that discovering eUCs is essential for enforcing entity and referential integrity but also optimizing queries on incomplete databases. However, discovering eUCs or eFDs is not an easy task. We have shown that the discovery problem of eUCs and the discovery problem of eFDs are each NP-complete and \( W[2]\)-complete in their arity, see Section 6.2 of Chapter 6 and Section 8.2 of Chapter 8, respectively. Given these likely intractability frontiers and potentially large search space in practice, we were still able to introduce row-efficient, column-efficient and hybrid discovery algorithms for both eUCs and eFDs, respectively in Chapter 6 and Chapter 8. Extensive experiments have been conducted on real-world benchmark datasets to test the discovery algorithms. In general, the row-efficient or the column-efficient algorithms perform best on datasets with a small number of columns or rows, respectively. On the other hand, the hybrid algorithms not only perform better but also scale well on datasets with large numbers of both rows and columns. In addition, we are able to compute semantic profiles for large datasets, which are known as Armstrong samples, by discovering eUCs and eFDs. As shown by our results, for large real-world datasets, sizes of Armstrong samples either for eUCs or eFDs, are typically under 1% relative to their original sizes. This demonstrates compelling evidence that Armstrong samples will be convenient for many data applications in management, reporting, and analytics. As a matter of fact, we demonstrated
9.1. Conclusion

a promising data profiling software which combines the discovery and semantic sampling algorithms of eUCs. The tool shows that cleaning dirty data and acquiring meaningful business rules are intrinsically linked. This was demonstrated on real-world scenarios (as demonstrated in Section 6.10). Another novel application has arisen from the discovery of eFDs, where we rank the relevance of the outputs by the number of redundant data values they cause within their scope. By inspecting the relevance of eFDs, the attention of users is typically drawn to meaningful business rules, or at least to eFDs that are more relevant to the given dataset. This holds because eFDs with higher ranks are supported by larger volumes of data redundancy in complete data fragments. In Chapter 7, we gave our vision for a general relational schema design framework that accommodates multiple data quality dimensions. To initiate work in this area, we defined the E-Redundancy Free Normal Form (E-RFNF), E-Boyce-Codd Normal Form (E-BCNF), and E-Third Normal Form (E-3NF) that enable users to tailor relational schema designs to the completeness and integrity requirements of applications. As an optimal result, we have shown how any given set of eUCs and eFDs can be transformed into a set of classical FDs such that E-BCNF and E-3NF can be computed by classical lossless BCNF-decomposition and classical lossless 3NF-synthesis algorithms, respectively. We have conducted comprehensive experiments on real-world benchmark schemata and data that show the effectiveness and efficiency of our proposed framework, but also quantify insights on data redundancy and classical normalization in unprecedented form. In particular, we gave in-depth demonstration on trade-offs between data redundancy and dependency preservation with real-world benchmark schemata. High data redundancy design offers better performance in queries but worse performance in updates. High dependency preservation keeps data intact by enforcing more meaningful rules. Different applications will have different emphasis on the two factors. Our framework is to provide quantified insights into trade-offs among incomplete database designs.
As a summary, embedded unique constraints and embedded functional dependencies provide a robust solution for the profiling of incomplete datasets as well as the design of database application schemata. The usability of data profiling with these constraints can establish more insight on bigger data, characterized by more column and rows, more missing data values, and more valid data dependencies. Our contributions in terms of covers, ranks, and semantic profiles enable users to explore the solution space of data profiling tools more effectively. Canonical covers reduce the search space without loss of information, ranks provide a total order of relevance on the solution space, and semantic profiles provide a user-friendly perspective of the output. Indeed, these tools may well enable users to identify business rules that have strong applications in data cleaning, query optimization, and database schema design. In fact, the combined class of eUCs and eFDs have made it possible for users to stipulate completeness and integrity requirements under one framework, capture rich sources of data redundancy that is independent of different interpretations of missing values, and tailor schema designs for incomplete data to data completeness requirements of applications. Hence, data under such schemata will be fit for purpose by design.

As an important side product of our work, we have been able to improve state-of-the-art on the discovery problem of classical FDs from data with missing values. Chapter 4 showed that our new hybrid algorithm improves the previously best algorithm in terms of runtime efficiency, column-, and row-scalability. Indeed, our technical contributions support a ratio that users can explore to balance the tradeoffs between runtime efficiency and memory consumption. The use of canonical over over LHS-reduced covers, the introduction of ranks for discovered FDs by the number of redundant data value occurrences, and semantic samples were further significant contributions to this classical problem. Furthermore, our experiments addressed different types of interpretations for missing values, which have a critical impact on real-world applications but have not been stressed sufficiently in
previous research. Our canonical covers, ranks, and semantic profiles make the outputs of FD discovery algorithms more useful, as they guide users to FDs that are more relevant for the given dataset and potentially the underlying application domain.

9.2 Future Work

There are multiple orthogonal areas that warrant future research.

The first area will introduce improvements over our current contributions. Examples are more efficient or more scalable variants for the algorithms that address the discovery problem, or the computation of Armstrong samples, the inclusion of more refined notions of covers [76] for sets of data dependencies, or the exploration of applications in data cleaning, integrity enforcement, and query optimization.

The second area includes the extension of our techniques to other classes of data dependencies. These may include tuple-generating dependencies such as multivalued dependencies [38] which capture redundant data value occurrences that cannot be expressed by functional dependencies, but also referential integrity constraints such as foreign keys and inclusion dependencies [23].

The third area includes the extension of our techniques to other data quality dimensions. So far, we have addressed the data integrity and data completeness dimensions, but other important dimensions include accuracy and timeliness.

The fourth area includes the extension of our techniques to other data models. These include the nested relational model, web models such as JSON [31], XML [109], or RDF [57], and uncertain data models such as probabilistic [42] or possibilistic [72] data models.
Bibliography


