Design Guidelines for Reducing Redundancy in Relational and XML Data

by

Solmaz Kolahi

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

Copyright © 2008 by Solmaz Kolahi
Abstract

Design Guidelines for Reducing Redundancy in Relational and XML Data

Solmaz Kolahi
Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto
2008

In this dissertation, we propose new design guidelines to reduce the amount of redundancy that databases carry. We use techniques from information theory to define a measure that evaluates a database design based on the worst possible redundancy carried in the instances. We then continue by revisiting the design problem of relational data with functional dependencies, and measure the lowest price, in terms of redundancy, that has to be paid to guarantee a dependency-preserving normalization for all schemas. We provide a formal justification for the Third Normal Form (3NF) by showing that we can achieve this lowest price by doing a good 3NF normalization.

We then study the design problem for XML documents that are views of relational data. We show that we can design a redundancy-free XML representation for some relational schemas while preserving all data dependencies. We present an algorithm for converting a relational schema to such an XML design.

We finally study the design problem for XML documents that are stored in relational databases. We look for XML design criteria that ensure a relational storage with low redundancy. First, we characterize XML designs that have a redundancy-free relational storage. Then we propose a restrictive condition for XML functional dependencies that guarantees a low redundancy for data values in the relational storage.
To my beloved family:

Vahid, Mom, Dad, and Siamak.
Acknowledgements

I would like to acknowledge the help and support of many people, who made the completion of this thesis possible. First and foremost, I would like to express my deep and sincere gratitude to my supervisor, Professor Leonid Libkin, for everything he taught me. His enthusiasm, broad knowledge, and insightful ideas have made working with him a great pleasure.

I would like to thank the members of my Ph.D. advisory committee, Professors Sheila McIlraith and Renée J. Miller for their helpful comments and advices over the years. I would also like to thank my external examiner, Professor Leopoldo Bertossi, and also Professors Michael Gruninger and John Mylopoulos for reading my thesis and providing useful feedbacks and enlightening questions. I am also grateful to Professor Alberto O. Mendelzon for being a great help in the early years of my studies before he sadly passed away. He will always be dearly remembered. I would like to acknowledge the financial support I received from the Department of Computer Science, University of Toronto, and the Government of Ontario.

I feel very lucky to have had the opportunity of working and learning among a great group of people in the database group at the University of Toronto. I would like to thank all my friends in the database group, especially Marcelo Arenas, for starting a very nice research, which was a great inspiration for me, and Ramona Truta for being a nice instructor and a good friend when I was her TA.

I would like to say a big thank you to all my Iranian friends in Toronto, who made me feel home in Canada. I will always remember the wonderful memories we shared. My special thanks goes to Afra Alishahi, Nilofar Arshadi, Elham Fazli, Afsaneh Fazly, and Nargol Rezvani for always listening to my complaints patiently, and Azadeh Farzan, Shiva Nejati, and Azadeh Shakery for being such great and inspiring friends ever since high school days.

I could have never finished my degree without the exceptional support of my loving
family. I would like to thank my dear husband, Vahid, for his technical support and feedbacks on my work over the years, for his patience and understanding during hard times, and more importantly, for being my best friend and companion. I am deeply indebted to my parents for their endless love, support, and encouragement in all aspects of my life, and for being my first and greatest teachers. I would also like to thank my dear brother, Siamak, for the happy childhood memories, and for always being such a caring and supportive friend.
Contents

1 Introduction ................................................. 1
  1.1 Summary of Contributions .......................... 4

2 Relational Databases - An Overview ...................... 6
  2.1 Database Schemas and Instances .................. 6
  2.2 Data Dependencies .................................. 9
    2.2.1 Functional and Key Dependencies .......... 9
    2.2.2 Multivalued and Join Dependencies ....... 11
    2.2.3 Equality-Generating Dependencies ....... 12
    2.2.4 Inclusion and Foreign Key Dependencies ... 13
  2.3 Designing Relational Data ......................... 14
    2.3.1 Normalizing Relational Data ............... 14

3 Information Theory and Normal Forms - An Overview .... 21
  3.1 Preliminaries ....................................... 22
    3.1.1 Schemas and Instances .................... 22
    3.1.2 Basics of Information Theory ............ 22
  3.2 Information Theory and Normal Forms ............... 23
    3.2.1 Relative Information Content ............ 24
    3.2.2 Justifying Perfect Normal Forms ........... 28
7 XML Design for Relational Storage

7.1 Introduction .................................................. 100
7.2 XML-to-Relational Mapping Scheme .......................... 102
  7.2.1 Translating XML Constraints ............................ 105
7.3 Relational Storage for Perfect XML Designs .................. 106
7.4 What is an Almost-Perfect Design for XML? .................. 108
  7.4.1 Guaranteed Information Content .......................... 111
7.5 A Different Mapping Scheme ................................. 114
7.6 Related Work ............................................... 118

8 Conclusions ................................................. 119

9 Future Work ............................................... 121

Bibliography ................................................. 123
## List of Figures

2.1 Movies Database. .......................................................... 7
2.2 An algorithm for computing the closure [AHV95]. .................. 10
2.3 An algorithm for producing BCNF schemas [AHV95]. .............. 17
2.4 A 3NF relation. ............................................................ 18
2.5 An algorithm for synthesizing 3NF schemas. ....................... 19
3.1 Calculating $\text{Ric}_k(p \mid \Sigma)$. ................................. 26
3.2 Information content vs. redundancy, where $\Sigma_1 = \{A \rightarrow C\}$ and $\Sigma_2 = \{A \rightarrow C, B \rightarrow C\}$. ....................... 27
4.1 An algorithm for synthesizing 3NF schemas [AHV95]. ............. 38
4.2 A database instance for the proofs of Propositions 4.5 and 4.10. .... 40
4.3 A database instance for the proof of Proposition 4.9. ............... 45
4.4 A database instance for the proof of Proposition 4.16. ............. 52
5.1 An XML document. ...................................................... 62
5.2 A DTD for the company database. ................................... 63
5.3 Tree representation of XML document. ............................... 64
5.4 A tree tuple. ............................................................. 73
5.5 An XML schema tree [HL03]. ......................................... 76
5.6 An XML data tree [HL03]. ............................................ 77
5.7 A school database [WT05]. ............................................ 78
5.8 XNF decomposition operations: (a) Moving attributes, (b) Creating new
element types [AL04]. ............................................................. 81
5.9 XNF decomposition algorithm [AL04]. ..................................... 82
5.10 A redundancy-free XML document. ........................................ 84

6.1 Conversion of relational data into a redundancy-free XML document. . 88
6.2 Dependency-preserving translation of relational data into redundancy-free
XML documents. ................................................................. 93
6.3 Finding the orderings of attributes using the algorithm in Section 6.3. . 95

7.1 An XML tree. ................................................................. 103
Chapter 1

Introduction

With the rapid growth and development of information systems, it becomes more essential to organize data in such a way that fast access and analysis of the data is feasible. Since the early 1970s, relational databases continue to serve as a popular framework for organizing and querying data.

In relational databases, data values are grouped into tables or relations, where each table has a number of related columns or attributes. The schema of a relational database consists of all the tables and their column names. Database design is the process of producing a good schema for a database by deciding how to group the columns of interest into tables.

There are several criteria for a good design [BPR88]: performance, or how fast the data is accessible; integrity, or to what extent the schema guarantees that the data is correct with respect to some constraints; understandability, or how coherent the structure of the database is to a user; and extensibility, or how easily the database can be extended to new applications. The most important factor in maintaining the integrity or correctness of a database is controlling the redundancy of data, which is the main concern of this thesis in the process of schema design.

Different methodologies exist to facilitate the process of database design. One is
starting with a conceptual model of the system, such as an *entity-relationship diagram (ERD)*, to capture all information entities and their relationships and then following the standard guidelines to convert the conceptual diagram into an initial design for a relational database. The result is a relational schema and a set of dependencies or integrity constraints that hold among data items.

Given a schema together with some data dependencies, *normalization* is the act of improving the schema by transforming it into a well-designed schema that is capable of representing the same information. A *normal form* specifies a set of syntactic conditions that a well-designed schema should satisfy. Normal forms usually deal with removing redundancies from a database to avoid possible anomalies during updates or insertions and to guarantee the integrity or correctness of the data. Codd defined the first normal form in the early 70s and since then database researchers defined a number of others, such as BCNF [Cod74], 3NF [Cod72], and 4NF [Fag77], that each disallows redundancies with respect to a different type of constraints.

In traditional normalization theory, a database is characterized as either redundant or non-redundant. However, between two databases that carry redundancy, one may be significantly worse than the other. Our goal is to provide richer guidelines that help a database designer choose a design that guarantees less redundancy. We use an information-theoretic tool that actually *measures* the amount of redundancy that a database allows. This tool was recently introduced by Arenas and Libkin [AL05], and was used to justify *perfect* designs that completely eliminate the possibility of redundancies.

Perfect designs, however, are not always the best choice for a number of reasons. First, it may not be possible to normalize a database into a perfect normal form without losing some of the integrity constraints. For instance, normalizing a relation schema into normal form BCNF may not be possible without losing some of the functional dependencies. Second, a normalization that completely eliminates redundancies may lead to producing too many relations, which will slow down the queries by requiring more joins.
In this dissertation, we introduce a measure for the highest redundancy found in instances to compare redundant designs and show that these designs can differ significantly, unlike what the traditional definition of redundancy shows. Our goal is to provide guidelines on how to choose the least-redundant design in case a non-redundant one is not achievable. We believe that this is an important choice, because the more redundant a database is, the more it is prone to inconsistencies after a series of insertions or updates.

In addition to relational databases, other databases can also suffer from poorly-designed data. With a recent shift to XML (eXtensible Markup Language) as the standard data model for exchange over the web, the problem of design and normalization has been revisited, and a number of normal forms have been proposed to avoid redundancies in XML documents. These normal forms are in the form of restrictive conditions on the structure of an XML document (the schema) and the data dependencies. Some of these documents are views of relational data. Our next goal is to present guidelines on how to provide a redundancy-free XML design for a relational database, even for schemas that do not have a non-redundant relational normalization that preserves all the constraints.

The works on XML design usually make an assumption of a native XML storage for reasoning about redundancies and anomalies. However, XML data is mostly stored in relations with the goal of using fast relational query engines. In this dissertation, we also look at the design problem for XML documents that are stored in relations. Our goal is to answer the following questions: how should we design an XML document to have a non-redundant relational storage, and in case this is not possible, how can we design a document to at least make sure that the redundancy of the relational storage is not too high. We believe it is important to answer these questions now before there is a standard update language for XML, in order to avoid having poorly-designed XML data in the near future that may lead to huge amounts of inconsistencies in the relational storage.
1.1 Summary of Contributions

With the goal of providing guidelines that help one choose a database design carrying less redundant data, we address several problems in this dissertation. We start by defining a measure that is capable of comparing relational designs containing functional dependencies based on the highest amount of redundancy they allow. This measure is called guaranteed information content and is defined using the information-theoretic framework developed by Arenas and Libkin [AL05]. Higher values of guaranteed information content correspond to better schemas that ensure less redundancy in instances.

We then calculate the least amount of redundancy that needs to be tolerated when preserving functional dependencies is a concern in normalization, and show how we can normalize a database into Third Normal Form (3NF) to guarantee this low redundancy and preserve dependencies at the same time. This can be a formal justification for 3NF, as a normal form that has been very popular in practical database design for years.

Along the way, we observe that relational schemas can differ significantly in terms of their guaranteed information content or the highest redundancy they allow. This means that a designer can decide whether or not a schema needs to be normalized based on how redundant the instances could be. We introduce a way of evaluating the quality of an arbitrary schema based on redundancy by calculating the highest amount of redundancy that may be carried by instances of the schema.

We continue our work by looking at the possibility of designing a non-redundant XML view of a relational schema with functional dependencies, which may not have a dependency-preserving non-redundant relational normalization. We characterize relational schemas for which such XML designs exist, and provide an algorithm that produces them.

Finally, we look at the problem of designing XML documents that are stored in relational database systems. We first show how XML functional dependencies get translated over the schema of the relational storage of an XML document. Then we extend the def-
inition of guaranteed information content for relational storage of XML, and show what conditions our XML design needs to satisfy in order to guarantee a high information content, or equivalently a low redundancy, in the relational storage.

We would like to mention that the results of this dissertation appeared in the following publications: the results of Chapter 4 appeared in [KL06], the results of Chapter 6 appeared in [Kol05, Kol07], and the results of Chapter 7 appeared in [KL07].
Chapter 2

Relational Databases - An Overview

In this chapter, we review basic notions of relational databases, including integrity constraints such as key and functional dependencies, and normalization theory for relational data.

2.1 Database Schemas and Instances

In relational databases, data is stored in relations or tables, where each row of a table intuitively corresponds to related pieces of information, called attributes, about a specific object or entity. The relational model was first introduced by Codd [Cod70] in the 70s, and has been very popular for storing and querying data in database systems ever since.

The database of Figure 2.1, for instance, consists of two relations Movies and Schedule, where Movies has four attributes for each movie: the title of the movie, the director of the movie, the actor who played in the movie, and the year it was produced. Relation Schedule has four attributes for each movie theater: the name and address of the theater, the title of a movie being shown there and the show time. Each such database has a structure, called the schema, which consists of the relation names and the attribute names, as well as a content, called the instance. We may also refer to rows and columns of an instance of a relation.
Formally speaking, we assume that we have a finite set of relation names as well as a finite set of attribute names, denoted by \textit{relname} and \textit{att} respectively. The domain or the set of possible values for each attribute \( A \in \textit{att} \) is denoted by \( \text{Dom}(A) \). A relation schema, written as \( R[U] \) consists of a relation name \( R \in \textit{relname} \) and a set \( U = \{A_1, \ldots, A_m\} \) of attribute names. We sometimes write \( R(A_1, \ldots, A_m) \) and refer to \( U \) as \textit{sort}(\( R \)). A \( U \)-tuple is a function with the domain \( U \) that assigns to each attribute \( A \in U \) a value in \( \text{Dom}(A) \). An instance \( I \) of a relation schema \( R[U] \) is a set of \( U \)-tuples.

In the relational database of Figure 2.1 for instance, we have a relation schema \( Movies[U] \), where \( U = \{\text{title}, \text{director}, \text{actor}, \text{year}\} \) and an instance \( I \) of six tuples, where for example the first tuple \( t_1 \in I \) can be described as follows: \( t_1(\text{title}) = \text{The Departed}, \ t_1(\text{director}) = \text{Scorsese}, \ t_1(\text{actor}) = \text{DiCaprio}, \ t_1(\text{year}) = 2006 \). We sometimes write such a tuple
as \(t_1\) (The Departed, Scorsese, DiCaprio, 2006). A database schema is a set of relation schemas \(S = \{R_1[U_1], \ldots, R_n[U_n]\}\). An instance \(I\) of a database schema \(S\) assigns an instance \(I(R)\) of \(U\)-tuples to each relation schema \(R[U] \in S\).

There are different paradigms for querying relational data [AHV95]. The first one is based on a set of algebraic operators, yielding a language called relational algebra, which has been of theoretical interest for years. Relational calculus, a variant of predicate calculus, is another paradigm that is based on writing queries as formulas in first-order logic. A third paradigm involves logic programming, and Datalog is the most well-known language in this paradigm.

Here we briefly present the main operators of relational algebra as they are extensively used in database design and normalization. The basic operators are: selection, projection, join, union, and difference, where the last two are the usual set-theoretic operators. The selection operator \(\sigma_c\) takes as input an instance \(I\) of relation schema \(R[U]\) and returns an instance of the same relation defined as \(\sigma_c(I) = \{t \in I \mid t \text{ satisfies condition } c\}\).

The projection operator is of the form \(\pi_X\), where \(X \subseteq U\) is a set of attribute names. It takes as input an instance \(I\) of \(R[U]\) and returns an instance of \(X\)-tuples defined as \(\pi_X(I) = \{t[X] \mid t \in I\}\). Here \(t[X]\) denotes an \(X\)-tuple obtained by restricting \(t\) to the attributes in \(X\).

Now let \(I, I'\) be instances of the two relation schemas \(R[U], R'[U']\) respectively and \(V = U \cup U'\). The join of \(I\) and \(I'\), written as \(I \bowtie I'\), is defined as the set \(\{s \mid s\ \text{is a } V\text{-tuple and there exist tuples } t \in I, t' \in I' \text{ such that } s[U] = t[U] \text{ and } s[U'] = t'[U']\}\).

Queries can be written in relational algebra by combining the basic operators. As an example, consider the database of Figure 2.1 and the following query: find theaters that show a movie in which Nicholson plays. This query can be expressed in relational algebra: \(\pi_{\text{theater}}(\sigma_{\text{actor}=\text{Nicholson}}(\text{Movies} \bowtie \text{Schedule}))\).
2.2 Data Dependencies

There are usually some restrictions on the values that can occur in a database instance. For example, one can specify a constraint for the database of Figure 2.1 that only one year is associated to each movie title. These semantic constraints are often specified by a set of data dependencies, denoted by $\Sigma$, using some language over the attributes of relation $R[U]$. In that case by schema we refer to both the relation and the constraints, written as $(R[U], \Sigma)$.

In normalizing relational data based on a set of dependencies, one often needs to reason about the data dependencies to have all the constraints inferred by them. This reasoning is referred to as the implication problem. Given any finite set $\Sigma \cup \{\varphi\}$ of data dependencies, implication is the problem of determining whether all databases satisfying $\Sigma$ also satisfy $\varphi$; this is usually written as $\Sigma \models \varphi$. The set of all dependencies implied by $\Sigma$ is denoted by $\Sigma^+$. For some dependencies, this set can be computed using a finite sound and complete set of inference rules, called axioms.

In this section we review some of the most common data dependencies defined for relational data: functional and key dependencies, inclusion and foreign key dependencies, equality-generating dependencies, and join and multivalued dependencies. We also present some of the known rules or algorithms that can be used for solving the implication problem for these dependencies.

2.2.1 Functional and Key Dependencies

Given a relation schema $R[U]$, a functional dependency (FD) is an expression of the form $X \rightarrow Y$, where $X, Y \subseteq U$. An instance $I$ of $R[U]$ satisfies $X \rightarrow Y$, written as $I \models X \rightarrow Y$, if for every two tuples $t_1, t_2 \in I$, $t_1[X] = t_2[X]$ implies $t_1[Y] = t_2[Y]$. For example, in relation Movies shown in Figure 2.1, no two tuples with the same title can have different years. This can be written as a functional dependency: $\text{title} \rightarrow \text{year}$. We
Input: A set $X$ of attributes and a set $\Sigma$ of functional dependencies.

Output: The closure $X^+$ under $\Sigma$.

unused := $\Sigma$;
closure := $X$;
repeat until no further change:
  if $W \rightarrow Z \in $ unused and $W \subseteq \text{closure}$ then
    unused := unused $-$ \{ $W \rightarrow Z$ \};
closure := closure $\cup$ $Z$;
return closure.

Figure 2.2: An algorithm for computing the closure [AHV95].

say that a functional dependency $X \rightarrow Y$ is trivial if $Y \subseteq X$. A key dependency is a functional dependency of the form $X \rightarrow U$. Then we say that $X$ is a superkey for the relation. If there is no superkey $Y$ such that $Y \subset X$ then we say that $X$ is a candidate key or just a key. For example, \{title, director, actor\} is a key for relation Movies.

Let $R[U]$ be a relation schema, $\Sigma$ be a set of FDs over $U$, and $X \subseteq U$ be a set of attribute names. The closure of $X$, written as $X^+$, is defined as the set of attributes $\{ A \mid \Sigma \models X \rightarrow A \}$. This set can be computed for any set of attributes $X$ and any set of FDs $\Sigma$ using the algorithm shown in Figure 2.2, which runs in quadratic time in the size of $U$. There is also a linear time algorithm [BB79, Ber79] for computing the closure. Knowing how to compute the closure in linear time, one can efficiently solve the implication problem since $\Sigma \models X \rightarrow Y$ if and only if $Y \subseteq X^+$.

The implication problem of functional dependencies can also be axiomatized. That is, there is a finite sound and complete set of inference rules that can be used to solve the implication problem. We now present an axiomatization for functional dependencies [Arm74]. In the following rules and hereafter, we write the union of two attribute sets $X$ and $Y$ as $XY$. 
FD1 (reflexivity): If $Y \subseteq X$, then $X \rightarrow Y$.

FD1 (augmentation): If $X \rightarrow Y$, then $XZ \rightarrowYZ$.

FD3 (transitivity): If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$.

2.2.2 Multivalued and Join Dependencies

A multivalued dependency (MVD) over a relation schema $R[U]$, is an expression of the form $X \multimap Y$, where $X, Y$ are subsets of $U$. An instance $I$ of $R[U]$ satisfies $X \multimap Y$ if for every two tuples $t_1, t_2 \in I$ such that $t_1[X] = t_2[X]$, there is another tuple $t_3 \in I$ such that $t_3[X] = t_1[X] = t_2[X]$, $t_3[Y] = t_1[Y]$, and $t_3[Z] = t_2[Z]$, where $Z = U - X - Y$.

For example, the instance of relation Movies shown if Figure 2.1 satisfies the multivalued dependency $title \multimap director$.

An algorithm, similar to the closure for functional dependencies, exists to solve the implication problem of MVDs [Bee80]. Since FDs and MVDs are usually considered together in the normalization of relational data, here we present a sound and complete set of rules that can be used to infer new dependencies from a set of MVDs and FDs defined over a relation $R[U]$ [BFH77]:

MVD0 (complementation): If $X \multimap Y$, then $X \multimap (U - X)$.

MVD1 (reflexivity): If $Y \subseteq X$, then $X \multimap Y$.

MVD2 (augmentation): If $X \multimap Y$, then $XZ \multimap YZ$.

MVD3 (transitivity): If $X \multimap Y$ and $Y \multimap Z$, then $X \multimap (Z - Y)$.

FMVD1 (conversion): If $X \rightarrow Y$, then $X \multimap Y$.

FMVD2 (interaction): If $X \multimap Y$ and $XY \rightarrow Z$, then $X \rightarrow (Z - Y)$.

It is also known that the set \{MVD0, \ldots, MVD3\} is an axiomatization for MVDs considered alone [BFH77, Men79].
Chapter 2. Relational Databases - An Overview

A join dependency (JD) over a relation schema \( R[U] \) is an expression of the form \( \Join [X_1, \ldots, X_n] \), where \( X_1 \cup \ldots \cup X_n = U \). An instance \( I \) of \( R[U] \) satisfies \( \Join [X_1, \ldots, X_n] \) if \( I = \pi_{X_1}(I) \Join \ldots \Join \pi_{X_n}(I) \). In other words, an instance satisfies a join dependency if it is equal to the join of some of its projections. This property is referred to as lossless join in the theory of normalization: information will not be lost if we decompose a relation into a number of smaller relations. We will discuss this concept later in Section 2.3.1. It is easy to observe that multivalued dependencies are special cases of join dependencies: \( X \rightarrow Y \) is equivalent to \( \Join [XY, X(U - XY)] \).

There is no axiomatization for the implication problem of JDs [AHV95]. However, there is a powerful tool, named chase, that was initially introduced [ABU79] to solve the implication problem of JDs from a set of FDs. It was later used to reason about other dependencies [MMS79]. Given a set of FDs and JDs, this tool requires exponential time and space to determine whether a JD can be inferred.

### 2.2.3 Equality-Generating Dependencies

Equality-generating dependencies (EGDs) were first introduced as a generalization for functional dependencies [BV84]. Given a database schema \( S = \{R_1[U_1], \ldots, R_n[U_n]\} \), a typed EGD is an expression of the form

\[
\forall (R_{i_1}(\bar{x}_1) \land \ldots \land R_{i_m}(\bar{x}_m) \rightarrow x = y),
\]

where \( \forall \) represents the universal closure of the formula, \( R_{i_1}, \ldots, R_{i_m} \) are relation names in \( S \), \( x, y \in \bar{x}_1 \cup \ldots \cup \bar{x}_m \), and there is an assignment of variables to columns such that each variable occurs only in one column, and \( x, y \) are assigned to the same column. If we do not have the last restriction on assigning variables to columns, then the dependency is untyped. Reasoning about EGDs can be done using the chase tool [AHV95].
2.2.4 Inclusion and Foreign Key Dependencies

Given a database schema \( S = \{R_1[U_1], \ldots, R_n[U_n]\} \), an inclusion dependency (ID) is an expression of the form \( R_i[X] \subseteq R_j[Y] \), where \( i, j \in [1, n] \), and \( X \subseteq U_i \) and \( Y \subseteq U_j \) are two sets of attributes of the same size. An instance \( I \) of \( S \) satisfies \( R_i[X] \subseteq R_j[Y] \) if for every tuple \( t \in I(R_i) \), there is a tuple \( t' \in I(R_j) \) such that \( t[X] = t'[Y] \). When \( Y \) is a superkey for relation \( R_j \), i.e., \( Y \rightarrow U_j \), the constraint is called a foreign key and is written as \( R_i[X] \subseteq_{FK} R_j[Y] \). As an example, consider the database schema consisting of two relations \( Movies \) and \( Schedule \) and the instance shown in Figure 2.1. This instance satisfies the inclusion dependency \( Schedule[movie] \subseteq Movies[title] \). Since \( title \) is not a key for relation \( Movies \), the foreign key \( Schedule[movie] \subseteq_{FK} Movies[title] \) does not hold in the instance.

There is a sound and complete set of rules to decide whether an inclusion dependency could be inferred from a set of other inclusion dependencies [CFP84]:

IND1 (reflexivity): \( R[X] \subseteq R[X] \).

IND2 (projection and permutation): If \( R[A_1, \ldots, A_m] \subseteq S[B_1, \ldots, B_m] \), then \( R[A_{i_1}, \ldots, A_{i_k}] \subseteq S[B_{i_1}, \ldots, B_{i_k}] \) for each sequence \( i_1, \ldots, i_k \) of distinct integers from \( \{1, \ldots, m\} \).

IND3 (transitivity): If \( R[X] \subseteq S[Y] \) and \( S[Y] \subseteq T[Z] \), then \( R[X] \subseteq T[Z] \).

There is not, however, an axiomatization for inclusion and functional dependencies considered together [CFP84]. In fact, the implication problem for IDs and FDs is known to be undecidable [CV85, Mit83]. Even by restricting the constraints to key and foreign key dependencies the implication problem still remains undecidable [FS00, FL01]. However, the implication problem of FDs and IDs is decidable in polynomial time when we deal only with unary IDs [CKV90] (the IDs of the form \( R[A] \subseteq S[B] \) with only one attribute on each side).
2.3 Designing Relational Data

To store data in relational databases, the first step is to decide what the set of attributes is and how these attributes should be grouped in tables. This step is usually referred to as schema design. There are different methodologies to facilitate this process in a standard way. One may want to start with a conceptual model of the system in order to capture the information needed for all entities in the system and the relationship between them. For instance, Entity-relationship diagrams (ERDs) provide a nice tool to model all the data items required for schema design and some integrity constraints at a conceptual level. There are also a number of guidelines, presented in many database textbooks [RG03, UGMW01, SKS06], on how to convert an ERD into an initial design for the schema of a relational database.

Given an initial database schema together with a set of data dependencies, schema refinement is the process of improving the schema by transforming it into another schema that has some nice properties. These nice properties are usually defined by a normal form, a set of syntactic conditions over the data dependencies. During this schema transformation, three criteria should be taken into account: preservation of data or losslessness, preservation of data dependencies, and achieving the properties defined by a certain normal form. We will formally define these criteria in the rest of this section, and review the most common normal forms for relational databases.

2.3.1 Normalizing Relational Data

The theory of normalization has a long history in the database research community. Codd [Cod72] showed for the first time that bad schema design may result in some anomalies when updating the database, and therefore we need to design schemas that satisfy certain criteria defined as normal forms. Consider for example the relation schema Schedule(theater, address, movie, time) and the instance of this relation depicted in Fig-
In this relation, we have a constraint that no two theaters with different addresses could have the same name. This can be stated as a functional dependency: \( \text{theater} \rightarrow \text{address} \). This dependency produces redundant data in the relation and therefore makes the database prone to anomalies during updates. For instance, if the address of the Varsity theater changes to 25 Bloor Street, all tuples concerning the Varsity theater should reflect this change. If any of these tuples does not get updated properly, the database will have inconsistent information caused by update anomaly. Codd [Cod72] introduced two normal forms to avoid this type of anomaly.

Other types of anomalies can also arise. Suppose that we want to insert a tuple for a new movie theater, but no movie is scheduled to be shown in the theater. This is an example of an insertion anomaly, because the schema is not appropriate to record this type of insertion. Now assume that movie The Departed is the only movie shown in the Cumberland theater, and that this show is going to be discontinued. By deleting the tuple (Cumberland, 100 Cumberland St., The Departed, 6pm), we will lose the information about the location of Cumberland theater. This is called a deletion anomaly, because deleting the movie has the effect of deleting the information about the theater and its address.

Different normal forms have been defined that provide guidelines to design databases that are less prone to anomalies. For instance, some normal forms, such as BCNF [Cod74] and 3NF [Cod72], define restrictive conditions on functional dependencies to avoid redundancies. Some normal forms deal with multivalued dependencies, such as 4NF [Fag77]; and some normal forms, such as PJ/NF and 5NF [Fag79, Vin97], consider join dependencies. There are also normal forms that eliminate redundancies caused by equality-generating dependencies [Wan05]. In this section, we review some of the well-known normal forms and present algorithms that transform a given database schema into another that satisfies a normal form. This process is called normalization.

More formally, given a database schema \( S = (R[U], \Sigma) \) and some normal form \( NF \), a
normalization algorithm produces another database schema, called an $\mathcal{NF}$-decomposition, $S' = \{(R_1[U_1], \Sigma_1), \ldots, (R_n[U_n], \Sigma_n)\}$ such that for every $i \in [1, n]$, $(R_i[U_i], \Sigma_i)$ satisfies normal form $\mathcal{NF}$. The decomposition is lossless [ABU79] if for every instance $I$ of $S$ there is an instance $I'$ of $S'$ such that

1. for every $i \in [1, n]$, $I'(R_i) = \pi_{U_i}(I)$, and
2. $I = I'(R_1) \Join \ldots \Join I'(R_n)$.

This property ensures that any instance of the original schema can be reconstructed by joining the instances of the decomposed schema, and therefore no tuple will be lost or generated during the normalization process. We say that $S'$ is a dependency-preserving decomposition of $S$ if $(\bigcup_{i=1}^{n} \Sigma_i)^+ = \Sigma^+$. That is, $\bigcup_{i=1}^{n} \Sigma_i$ and $\Sigma$ are equivalent.

**Boyce-Codd Normal Form (BCNF)**

Boyce-Codd normal form (BCNF) was introduced by Boyce and Codd [Cod74] to avoid all update anomalies caused by functional dependencies. Let $R[U]$ be a relation schema and $\Sigma$ be a set of functional dependencies. Then $(R[U], \Sigma)$ is in BCNF if for every nontrivial functional dependency $X \rightarrow Y \in \Sigma^+$, $X$ is a superkey. A database schema $S$ is in BCNF if every relation in $S$ is in BCNF.

Consider, for example, the relation schema $\text{Movies}(\text{title, director, actor, year})$ and the set of functional dependencies $\Sigma = \{\text{title} \rightarrow \text{year}\}$. This schema is not in BCNF since in the nontrivial functional dependency $\text{title} \rightarrow \text{year}$, $\text{title}$ is not a superkey. To check whether a schema $(R[U], \Sigma)$ is in BCNF, it is sufficient to check whether for every functional dependency $X \rightarrow Y \in \Sigma$, we have $X \rightarrow U$. This can be efficiently done using the linear time algorithm for the implication of functional dependencies [BB79, Ber79].

Given a schema $(R[U], \Sigma)$ that does not satisfy BCNF, one can produce a lossless BCNF decomposition of $(R[U], \Sigma)$, using the algorithm shown in Figure 2.3. In this algorithm, $\pi_X(\Sigma)$ represents the projection of a set of functional dependencies $\Sigma$ on
Input: A relation schema \((R[U], \Sigma)\), where \(\Sigma\) is a set of functional dependencies.

Output: A database schema \(S\) in BCNF.

\[ S := \{(R[U], \Sigma)\}; \]

repeat until \(S\) is in BCNF:

Choose a relation schema \((R'[U'], \Sigma')\) in \(S\) that is not in BCNF;

Choose nonempty disjoint sets \(X, Y, Z \subset U\) such that

\[ X \cup Y \cup Z = U, \Sigma' \models X \rightarrow Y, \text{ and } \Sigma' \nmodels X \rightarrow A \text{ for every } A \in Z; \]

Replace \((R'[U'], \Sigma')\) in \(S\) by \((R_1[XY], \pi_{XY}(\Sigma'))\) and \((R_2[XZ], \pi_{XZ}(\Sigma'))\), where \(R_1, R_2\) are fresh relation names;

return \(S\).

Figure 2.3: An algorithm for producing BCNF schemas [AHV95].

a set of attributes \(X\). This set is defined as \(\{Y \rightarrow Z \mid \Sigma \models Y \rightarrow Z, \text{ and } YZ \subseteq X\}\). Consider, for example, relation schema \((ABC, \{A \rightarrow B, B \rightarrow C\})\). Applying the BCNF decomposition algorithm to this schema will output schema \(S = \{(BC, \{B \rightarrow C\}), (AB, \{A \rightarrow B\})\}\), which is a dependency-preserving decomposition.

It is not always possible to produce a dependency-preserving BCNF decomposition for a relation schema. For instance, by looking at all possible lossless decompositions of \((ABC, \{AB \rightarrow C, C \rightarrow B\})\), we see that this schema does not admit a dependency-preserving BCNF decomposition.

Third Normal Form (3NF)

Third normal form (3NF) was introduced by Codd [Cod72] to avoid some update anomalies caused by functional dependencies. Let \(R[U]\) be a relation schema and \(\Sigma\) be a set of functional dependencies. We say that an attribute \(A \in U\) is prime if it is an element of some key of \(R[U]\). A schema \((R[U], \Sigma)\) is in 3NF if for every nontrivial functional dependency \(X \rightarrow A \in \Sigma^+\), \(X\) is a superkey or \(A\) is prime. We say that a database
schema $S$ is in 3NF if every relation schema in $S$ is in 3NF.

Note that 3NF is less restrictive than BCNF and hence allows some update anomalies. Consider for example relation $\text{Teach}(\text{instructor}, \text{course}, \text{time})$ and FDs $\text{instructor, time} \rightarrow \text{course}, \text{course} \rightarrow \text{instructor}$. This schema is in 3NF, but it is not in BCNF. Now consider an instance of this relation shown in Figure 2.4. The fact that the course Databases is taught by Smith is repeated in two tuples, and this redundancy may cause anomalies when changing the instructor of a course.

Checking whether a schema is in 3NF is NP-complete in general [JF82]. However, most database schemas in real life have some nice properties that make this check less expensive. There have been some old [MR89] and recent attempts [GPW06] to present polynomial algorithms for checking whether an attribute is prime or a schema is in 3NF, given that the schema satisfies some conditions.

For every database schema $S$ that does not satisfy 3NF, there is another schema $S'$ in 3NF that is a lossless dependency-preserving decomposition of $S$. One way to generate this schema is by using the synthesis algorithm introduced by Bernstein [Ber76] and further extended by Biskup et al. [BDB79] that is shown in Figure 2.5. There have been different attempts to improve the original algorithm with the goal of achieving a smaller schema, mostly in terms of the number of relations in the produced schema [LTK81, Zan82].

We now define some terminology needed to understand the algorithm in Figure 2.5. Given a set of functional dependencies $\Sigma$, a minimal cover of $\Sigma$ is a set of functional dependencies $\Sigma'$ such that $\Sigma'$ is a minimal set of dependencies that implies $\Sigma$. The algorithm in Figure 2.5 constructs a minimal cover of the functional dependencies in the input relation.
Input: A relation schema \((R[U], \Sigma)\), where \(\Sigma\) is a set of functional dependencies.

Output: A database schema \(S\) in 3NF.

find a minimal cover \(\Sigma'\) of \(\Sigma\);

partition \(\Sigma'\) into groups \(\Sigma'_1, \ldots, \Sigma'_n\) such that all FDs in each group have identical left-hand sides;

\[ S := \{(R_i[U_i], \Sigma'_i) \mid U_i \text{ is the set of all attributes appearing in } \Sigma'_i\}; \]

if there is no \((R_i[U_i], \Sigma'_i)\) such that \(U_i\) is a superkey for \(R[U]\) then

choose a key \(X\) of \(R[U]\);

\[ S := S \cup \{(R_{n+1}[X], \emptyset)\}; \]

return \(S\).

Figure 2.5: An algorithm for synthesizing 3NF schemas.

dependencies \(\Sigma'\) such that:

1. each dependency in \(\Sigma'\) is of the form \(X \rightarrow A\), where \(A\) is an attribute;

2. \(\Sigma\) is equivalent to \(\Sigma'\), i.e., \(\Sigma^+ = \Sigma'^+\);

3. no proper subset of \(\Sigma'\) is equivalent to \(\Sigma\); and

4. for each dependency \(X \rightarrow A \in \Sigma'\), there is no \(Y \subsetneq X\) such that \(\Sigma \models Y \rightarrow A\).

Consider for example relation schema \((ABCD, \{A \rightarrow B, A \rightarrow C\})\). Applying the 3NF synthesis algorithm to this schema can result in a 3NF database schema consisting of two relation schemas: \((ABC, \{A \rightarrow B, A \rightarrow C\})\) and \((AD, \emptyset)\). Note that adding the second relation according to the last step of the algorithm ensures that we produce a lossless decomposition of the original schema.

Fourth Normal Form (4NF)

Fourth Normal Form (4NF) was introduced by Fagin [Fag77] to eliminate redundancies that are caused by multivalued dependencies. To see an example of such a redundancy,
consider the instance of relation *Movies* shown in Figure 2.1 that satisfies the MVD $\text{title} \rightarrow \rightarrow \text{director}$. The actor Myers is assigned to the movie Shrek the Third in two different tuples, because the movie has two directors, and all the other information should be repeated for each director.

Let $R[U]$ be a relation schema and $\Sigma$ be a set of functional and multivalued dependencies. Then $(R[U], \Sigma)$ is in 4NF if for every nontrivial MVD $X \rightarrow \rightarrow Y \in \Sigma^+$, $X$ is a superkey for relation $R$. A database schema $S$ is in 4NF if every relation schema in $S$ is in 4NF. A decomposition algorithm producing 4NF schemas can be developed by slightly modifying the BCNF decomposition algorithm, which is not necessarily dependency-preserving [AHV95].
Chapter 3

Information Theory and Normal Forms - An Overview

The problem of database normalization is one of the oldest and most researched in database theory and practice. The descriptions of well-known normal forms, such as 3NF and BCNF, appears in practically all texts (see, e.g., [AHV95, KBL06, LLL99]), and many practical tools exist for database design. Nonetheless, to answer what makes a database design good, texts typically offer a rather informal explanation based on the absence of update anomalies or elimination of redundancies. Papers [Fag79, Fag81, LV00] that attempt a more formal evaluation of normal forms still appeal to the notions of eliminating update anomalies. A rather recent approach was taken by Arenas and Libkin [AL05] that uses information theory to evaluate both relational and XML designs. This approach is independent of any query or update language, and evaluates the design of a database based on the schema and the constraints.

In this chapter, we review the basic notions of the information-theoretic framework and present the main results by Arenas and Libkin [AL05]. We will use this framework in Chapters 4 and 7 to establish new results on the design problem for relational and XML data.
3.1 Preliminaries

In this section, we present the definitions and concepts needed for applying information theory in database normalization.

3.1.1 Schemas and Instances

In this chapter, we assume that elements of database instances come from a countably-infinite domain; to be concrete, we assume it to be \( \mathbb{N}^+ \), the set of positive integers. Therefore, an instance \( I \) of a database schema \( S \) assigns to each \( m \)-attribute relation \( R \) in \( S \) a finite subset \( I(R) \) of \( \mathbb{N}^+^m \). We let \( \text{adom}(I) \) stand for the active domain of \( I \): the set of all elements of \( \mathbb{N}^+ \) that occur in \( I \). The size of \( I(R) \) is defined as \( \|I(R)\| = |\text{sort}(R)| \cdot |I(R)| \), and the size of \( I \) is \( \|I\| = \sum_{R \in S} \|I(R)\| \). The set of positions in \( I \), denoted by \( \text{Pos}(I) \), is defined as the set \( \{(R,t,A) \mid R \in S, t \in I(R) \text{ and } A \in \text{sort}(R)\} \).

We shall deal with integrity constraints that are expressed in the form of some data dependencies for schemas, and we refer to schemas \( (S, \Sigma) \), where \( S \) is a set of relation names and \( \Sigma \) is a set of constraints. We let \( \text{inst}(S, \Sigma) \) stand for the set of all instances of \( S \) satisfying \( \Sigma \). We write \( \text{inst}_k(S, \Sigma) \) for the set of instances \( I \in \text{inst}(S, \Sigma) \) with \( \text{adom}(I) \subseteq [1,k] \).

3.1.2 Basics of Information Theory

Entropy is a fundamental concept in information theory that is defined to measure the amount of information provided by a certain event. Assume that an event can have \( n \) different outcomes \( s_1, \ldots, s_n \), each with probability \( p_i, i \in [1,n] \), and we would like to know how much information is gained by knowing that \( s_i \) has occurred. We need a surprise function \( h : [0,1] \to \mathbb{R} \) that given a probability of a certain outcome returns the amount of surprise in knowing that the outcome has occurred. Function \( h \) should be a continuous monotone function on the probability space such that \( h(1) = 0 \) (no
surprise), and \( h(0) = \infty \) (infinite surprise). Furthermore, for two independent events \( s_i, s_j \), we should have \( h(p(s_i, s_j)) = h(p_i \cdot p_j) = h(p_i) + h(p_j) \). The only functions that have these properties are functions of the form \( h(p) = -k \log_b(p) \), where \( k, b > 0 \) [Sha48]. By convention, we take \( k = 1 \) and \( b = 2 \).

The entropy of a probability distribution is then the average amount of information that can be gained by knowing that a certain event has occurred. For a probability space \( \mathcal{A} = (\{s_1, \ldots, s_n\}, P_A) \), where \( P_A \) is a probability distribution, its entropy is defined as

\[
H(\mathcal{A}) = \sum_{i=1}^{n} P_A(s_i) \log \frac{1}{P_A(s_i)}.
\]

For probabilities that are zero, we adopt the convention that \( 0 \log \frac{1}{0} = 0 \), since we have \( \lim_{x \to 0} x \log \frac{1}{x} = 0 \). It is known that \( 0 \leq H(\mathcal{A}) \leq \log n \), with \( H(\mathcal{A}) = \log n \) only for the uniform distribution \( P_A(s_i) = 1/n \) [CT91].

We shall also need the concept of conditional entropy. For two probability spaces \( \mathcal{A} = (\{s_1, \ldots, s_n\}, P_A) \), \( \mathcal{B} = (\{s'_1, \ldots, s'_m\}, P_B) \), and probabilities \( P(s'_j, s_i) \) of all the events \( (s'_j, s_i) \) (\( P_A \) and \( P_B \) may not be independent), the conditional entropy of \( \mathcal{B} \) given \( \mathcal{A} \), denoted by \( H(\mathcal{B} | \mathcal{A}) \), gives the average amount of information provided by \( \mathcal{B} \) if \( \mathcal{A} \) is known [CT91]. If \( P(s'_j | s_i) = P(s'_j, s_i)/P_A(s_i) \) are conditional probabilities, then

\[
H(\mathcal{B} | \mathcal{A}) = \sum_{i=1}^{n} \left( P_A(s_i) \sum_{j=1}^{m} P(s'_j | s_i) \log \frac{1}{P(s'_j | s_i)} \right).
\]

### 3.2 Information Theory and Normal Forms

An information-theoretic framework was recently proposed by Arenas and Libkin [AL05] that is used to justify relational normal forms and to provide a test of “goodness” of normal forms for other data models. This framework is completely independent of the notions of update or query languages, and is based on the intrinsic properties of the data. Unlike previously proposed information-theoretic measures [Lee87, CP87, DR00, LL03], this measure takes into account both data and schema constraints.
Given a database schema $S$, a set of constraints $\Sigma$, and an instance $I$ of $(S, \Sigma)$, the information-theoretic measure assigns a number to every position $p$ in the instance that contains a data value, by calculating a conditional entropy of a certain probability distribution and then normalizing to the interval $[0, 1]$. This number, which is called relative information content with respect to constraints $\Sigma$ and is written as $\text{Ric}_I(p \mid \Sigma)$, ranges between 0 and 1 and shows how much redundancy is carried by position $p$. Intuitively, if $\text{Ric}_I(p \mid \Sigma) = 1$, then $p$ carries the maximum possible amount of information: nothing about it can be inferred from the rest of the instance. Smaller values of $\text{Ric}_I(p \mid \Sigma)$ show that positions carry some amount of redundancy, as some information about them can be inferred. Next we give a formal definition of this measure.

### 3.2.1 Relative Information Content

Fix a schema $S$ and a set $\Sigma$ of constraints, and let $I \in \text{inst}(S, \Sigma)$ with $\|I\| = n$. Recall that the set of positions in $I$, denoted by $\text{Pos}(I)$, is defined as the set $\{(R, t, A) \mid R \in S, t \in I(R) \text{ and } A \in \text{sort}(R)\}$. We now want to define $\text{Ric}_I(p \mid \Sigma)$, the relative information content of a position $p \in \text{Pos}(I)$ with respect to the set of constraints $\Sigma$. We want this value to be normalized to the interval $[0, 1]$. Since the maximum value of entropy for a discrete distribution on $k$ elements is $\log k$, we shall define, for all $k$, a measure $\text{Ric}_I^k(p \mid \Sigma)$ that works for instances $I \in \text{inst}_k(S, \Sigma)$, and take the limit of the ratio $\frac{\text{Ric}_I^k(p \mid \Sigma)}{\log k}$ as $k \to \infty$.

Since this is a measure of the amount of redundancy, intuitively, we want to measure how much, on average, the value of position $p$ is determined by any set of positions in $I$. For that, we take a set $X \subseteq \text{Pos}(I) - \{p\}$ and assume that the values in those positions $X$ are lost, and then someone restores them from $[1, k]$. Then, we measure how much information about the value in $p$ this provides by calculating the entropy of a suitably chosen distribution. The average such measure is $\text{Ric}_I^k(p \mid \Sigma)$.

Formally, we assume that $I$ has $n$ positions (which we enumerate as $1, \ldots, n$), and
fix an \( n \)-element set of variables \( \{ v_i \mid 1 \leq i \leq n \} \). Fix a position \( p \in \text{Pos}(I) \), and let \( \Omega(I,p) \) be the set of all \( 2^n-1 \) vectors \((a_1, \ldots, a_{p-1}, a_{p+1}, \ldots, a_n)\) such that for every \( i \in [1,n]-\{p\}, a_i \) is either \( v_i \) or the value in the \( i \)-th position of \( I \). We make this into a probability space \( \mathcal{A}(I,p) = (\Omega(I,p), P_u) \) with the uniform distribution \( P_u(\bar{a}) = 2^{1-n} \).

We next define conditional probabilities \(^1 P_k(a \mid \bar{a}) \) that show how likely \( a \) is to occur in position \( p \), if values are removed from \( I \) according to the tuple \( \bar{a} \in \Omega(I,p) \). Let \( I(a,\bar{a}) \) be obtained from \( I \) by putting \( a \) in position \( p \), and \( a_i \) in position \( i \neq p \). A substitution is a map \( \sigma : \bar{a} \rightarrow [1,k] \) that assigns a value to each \( a_i \) which is a variable, and leaves other \( a_i \)'s intact. We let \( \text{SAT}_k^\Sigma(I(a,\bar{a})) \) be the set of all substitutions \( \sigma \) such that \( \sigma(I(a,\bar{a})) \models \Sigma \) and \( |\sigma(I(a,\bar{a}))| = |I| \) (the latter ensures that no two tuples collapse as the result of applying \( \sigma \)). Then \( P_k(a \mid \bar{a}) \) is defined as:

\[
P_k(a \mid \bar{a}) = \frac{\text{SAT}_k^\Sigma(I(a,\bar{a}))}{\sum_{b \in [1,k]} \text{SAT}_k^\Sigma(I(b,\bar{a}))}.
\]

With this, we define \( \text{Ric}_I^k(p \mid \Sigma) \) as

\[
\sum_{\bar{a} \in \Omega(I,p)} \left( \frac{1}{2^{n-1}} \sum_{a \in [1,k]} P_k(a \mid \bar{a}) \log \frac{1}{P_k(a \mid \bar{a})} \right).
\]

Since \( \sum_{a \in [1,k]} P_k(a \mid \bar{a}) \log \frac{1}{P_k(a \mid \bar{a})} \) measures the amount of information in \( p \), given constraints \( \Sigma \) and some missing values in \( I \), represented by the variables in \( \bar{a} \), our measure \( \text{Ric}_I^k(p \mid \Sigma) \) is the average such amount over all \( \bar{a} \in \Omega(I,p) \).

**Example 3.1** Consider relation \( R(A, B, C) \) with the set of FDs \( \Sigma = \{ A \rightarrow B \} \), and an instance \( I \in \text{inst}(R, \Sigma) \) shown in Figure 3.1. Let \( p \) denote the position of the gray cell in the instance. The figure shows some steps that lead to calculating \( \text{Ric}_I^k(p \mid \Sigma) \), when \( k = 7 \). \( \Box \)

\(^1\)Technically, we should refer not to \( P_k \) but rather \( P_{I,\Sigma,k} \) but \( I \) and \( \Sigma \) will always be clear from the context.
To see that $\text{Ric}_I^k(p \mid \Sigma)$ is a conditional entropy, we define a probability distribution on $[1, k]$ as follows:

$$P'_k(a) = \frac{1}{2^{n-1}} \sum_{\bar{a} \in \Omega(I, p)} P_k(a \mid \bar{a}).$$

Intuitively, this probability shows how likely an element from $[1, k]$ is to satisfy $\Sigma$ when put in position $p$, given all possible interactions between $p$ and sets of positions in $I$. If $\mathcal{B}_\Sigma^k(I, p)$ is the probability space $([1, k], P'_k)$, then $\text{Ric}_I^k(p \mid \Sigma)$ is the conditional entropy:

$$\text{Ric}_I^k(p \mid \Sigma) = H(\mathcal{B}_\Sigma^k(I, p) \mid \mathcal{A}(I, p)).$$

Since the domain of $\mathcal{B}_\Sigma^k(I, p)$ is $[1, k]$, we have $0 \leq \text{Ric}_I^k(p \mid \Sigma) \leq \log k$. To normalize this, we consider the ratio $\text{Ric}_I^k(p \mid \Sigma)/\log k$. The key observation of Arenas...
and Libkin [AL05] is that for most reasonable constraints \( \Sigma \) (certainly for all constraints definable in first-order logic), this sequence converges as \( k \to \infty \), and we thus define

\[
Ric_I(p \mid \Sigma) = \lim_{k \to \infty} \frac{Ric^k_I(p \mid \Sigma)}{\log k}.
\]

If \( Ric_I(p \mid \Sigma) = 1 \), the information carried by position \( p \) is at a maximum, and there is no redundancy in position \( p \). If the data in position \( p \) is redundant and the value in this position can be inferred from the rest of the instance and constraints, then \( Ric_I(p \mid \Sigma) \) gets a value in \([0, 1)\) to show how redundant the value in position \( p \) is.

**Example 3.2** Consider relation \( R(A, B, C, D) \), two sets of FDs \( \Sigma_1 = \{A \rightarrow C\} \) and \( \Sigma_2 = \{A \rightarrow C, B \rightarrow C\} \), and three instances \( I_1, I_2, \) and \( I_3 \) in Figure 3.2 that are in both \( \text{inst}(R, \Sigma_1) \) and \( \text{inst}(R, \Sigma_2) \). Let \( p_1, p_2, p_3 \) denote the position of the gray cells in the instances. We observe that the information content of the gray cell decreases as it becomes more redundant by adding tuples. We also see how the information content changes by adding more constraints that make the gray cell even more redundant. \( \square \)
3.2.2 Justifying Perfect Normal Forms

Ideally, we want databases in which every position carries the maximum amount of information. The notion of being well-designed is accordingly defined as follows [AL05]:

**Definition 3.3** A database schema $S$ with a set of constraints $\Sigma$ is well-designed if for every instance $I \in \text{inst}(S, \Sigma)$ and every position $p \in \text{Pos}(I)$, $\text{Ric}_I(p \mid \Sigma) = 1$.

In other words, well-designed databases are the ones that allow absolutely no redundancy in any position. It is known [AL05] that this definition corresponds exactly to the definition of having no redundancy by Vincent [Vin99], which calls a data value $v$ in instance $I$ redundant if replacing $v$ with any other value would violate the constraints. Using the notion of being well-designed, well-known normal forms, such as BCNF and 4NF, have been justified, and the corresponding normalization algorithms have been proved to always produce a well-designed database.

Although some of these normal forms were previously justified by showing that they eliminate the possibility of redundancy or update anomalies [BG80, LJ82, Fag81, Vin99], the information-theoretic technique provided the first justification for each step of the normalization algorithms [AL05] by showing that these steps never decrease the amount of information content in any position of an instance. Furthermore, since the information-theoretic framework enables us to measure the amount of redundancy in databases that are not well-designed, it can be used to justify normal forms, such as 3NF, that do not completely eliminate redundancies, as we will see in Chapter 4. Now we briefly present some of the known results, and refer the reader to other references [AL05, Are05] for more details.

We consider schemas that contain join dependencies. A join dependency over a relation schema $R[U]$ can be written as a first-order sentence of the form:

$$\forall(R(\bar{x}_1) \land \ldots \land R(\bar{x}_m) \rightarrow R(\bar{x})).$$
where $\forall$ represents the universal closure of the formula, $\bar{x} \subseteq \bar{x}_1 \cup \ldots \cup \bar{x}_m$, every variable not in $\bar{x}$ occurs precisely in one $\bar{x}_i$ ($i \in [1, m]$), and there is an assignment of variables to columns such that each variable occurs in exactly one column. An important result characterizes well-designed schemas that contain functional and join dependencies:

**Theorem 3.4 (Arenas and Libkin [AL05])** Let $\Sigma$ be a set of FDs and JDs over a database schema $S$. Then $(S, \Sigma)$ is well-designed if and only if for every relation $R \in S$ and every nontrivial join dependency $\forall (R(\bar{x}_1) \land \ldots \land R(\bar{x}_m) \rightarrow R(\bar{x}))$ in $\Sigma^+$, there exists $M \subseteq \{1, \ldots, m\}$ such that:

1. $\bar{x} \subseteq \bigcup_{i \in M} \bar{x}_i$, and
2. for every $i, j \in M$, $\forall (R(\bar{x}_1) \land \ldots \land R(\bar{x}_m) \rightarrow \bar{x}_i = \bar{x}_j)$ is in $\Sigma^+$.

By restricting Theorem 3.4 to the case of multivalued and functional dependencies, we obtain an information-theoretic justification for normal forms 4NF and BCNF:

**Theorem 3.5 (Arenas and Libkin [AL05])** Let $\Sigma$ be a set of integrity constraints over a database schema $S$.

1. If $\Sigma$ contains only FDs and MVDs, then $(S, \Sigma)$ is well-designed if and only if it is in 4NF.
2. If $\Sigma$ contains only FDs, then $(S, \Sigma)$ is well-designed if and only if it is in BCNF.

The information-theoretic measure is also capable of justifying the steps in a normalization algorithm at the instance level. Here, we present a justification [AL05] for the BCNF decomposition algorithm (see Section 2.3.1). In each step of this algorithm, we choose a relation $(R[U], \Sigma)$ in database schema $S$ that is not in BCNF and disjoint sets of attributes $X, Y, Z \subseteq U$ such that $XYZ = U$, $X \rightarrow Y \in \Sigma^+$, and $X \not\rightarrow A$ for every $A \in Z$. Then every instance $I$ of $R$ gets decomposed into $I_{XY}$ with FDs $\pi_{XY}(\Sigma)$ and $I_{XZ}$ with FDs $\pi_{XZ}(\Sigma)$. To compare the amount of information content in positions of
instance $I$ and its decomposition $I_{XY}, I_{XZ}$, we need to somehow associate the positions in $I$ and the positions in the decomposition. To do this in a natural way, we define two partial maps $\pi_1 : Pos(I) \to Pos(I_{XY})$ and $\pi_2 : Pos(I) \to Pos(I_{XZ})$. If $p = (R, t, A)$ for some attribute $A \in XY$ and some tuple $t \in I$, then $\pi_1(p)$ is defined and equals the position of $\pi_{XY}(t)[A]$ in $I_{XY}$. The mapping $\pi_2$ is defined similarly.

The following theorem shows that the BCNF decomposition algorithm does not decrease the amount of information content:

**Theorem 3.6 (Arenas and Libkin [AL05])** Let $(X, Y, Z)$ partitions the attributes of $R$, and let $X \rightarrow Y \in \Sigma^+$. Let $I \in inst(R, \Sigma)$ and $p \in Pos(I)$. Then $Ric_{I}(p \mid \Sigma) \leq Ric_{I_{XY}}(\pi_1(p) \mid \Sigma_{XY})$ and $Ric_{I}(p \mid \Sigma) \leq Ric_{I_{XZ}}(\pi_2(p) \mid \Sigma_{XZ})$.

We say that a decomposition algorithm is *effective* in instance $I$ if for one of its basic steps and for some position $p$, the inequality of the above theorem is strict. That is, the information content of some position increases. This leads to another characterization of BCNF:

**Proposition 3.7 (Arenas and Libkin [AL05])** A schema $(R, \Sigma)$ is in BCNF if and only if no decomposition algorithm is effective in $(R, \Sigma)$.

The proposition also explains, from the information-theoretic point of view, why the decomposition algorithm of Section 2.3.1 should stop right after the schema is in BCNF.
Chapter 4

Characterizing Almost-Perfect Normal Forms

An information-theoretic framework, which was presented in Chapter 3, was recently used to justify perfect normal forms like BCNF that completely eliminate redundancies. The main notion is that of an information content of each datum in an instance (which is a number in \([0, 1]\)): the closer to 1, the less redundancy it carries. In practice, however, one usually settles for 3NF which, unlike BCNF, may not perfectly eliminate all redundancies but always guarantees dependency preservation.

Our first goal in this chapter is to use the information-theoretic framework to evaluate non-perfect normal forms that tolerate some redundancy in order to achieve dependency preservation. For each such normal form, we define the price of dependency preservation as an information-theoretic measure of minimum redundancy that gets introduced to compensate for dependency preservation. We show that a 3NF normalization achieves the best possible price. Moreover, we compare normal forms based on the highest amount of redundancy or, equivalently, the lowest amount of information content that they allow in a single position of a database as well as in the entire database on average. Based on these studies, the goal is to propose guidelines on how to choose the best design if
dependency preservation and redundancy elimination are both of concern. Finally, we introduce a way of evaluating the quality of a schema, based on the highest amount of redundancy that it allows. The results of this chapter were partly presented in [KL06].

4.1 Introduction

In this chapter, we provide a justification for one of the most popular and commonly-used normal forms, 3NF. We adopt the information-theoretic framework that was introduced by Arenas and Libkin [AL05] and presented in Chapter 3 to reason about databases. In Chapter 3, we presented known results [AL05] that justify perfect normal forms, normal forms that do not allow redundancy whatsoever, using the notion of being well-designed. In every instance $I$ of a well-designed schema, the relative information content $R_{IC_{I}}(p | \Sigma)$ of every position $p$ is 1. That is, no redundancies are allowed. Characterizations of well-designed normal forms for different types of dependencies were obtained [AL05]: for example, if $\Sigma$ consists only of functional dependencies, then being well-designed is the same as being in the Boyce-Codd normal form (BCNF).

While this does justify a normal form that is perhaps the most popular one for database texts, BCNF is not the most common and popular normal form in practice – that role belongs to 3NF. Practical database design tips (e.g., in books [GSS07, SP02, Dew06]) usually refer to a “normalized” database schema as a schema that satisfies 3NF.

The main property possessed by 3NF, but not by BCNF is dependency preservation: for every schema, there always exists a lossless decomposition into 3NF that preserves all the functional dependencies. Throughout this chapter, by a dependency-preserving decomposition, we mean that the set of FDs on the original schema is equivalent to the set of projected FDs on the decomposed schemas. This is a very important property for integrity enforcement, as DBMSs provide a variety of mechanisms to ensure that integrity constraints are enforced during updates. Keeping the constraints in the form
of functional dependencies makes the integrity enforcement much faster since enforcing FDs does not require joins across different relations.

Notice that it is not always possible to do a dependency-preserving BCNF normalization to achieve a well-designed schema. For instance, for relation $R(A, B, C)$ and set of FDs $\Sigma = \{AB \rightarrow C, C \rightarrow B\}$, any BCNF decomposition will lose some functional dependencies, and hence a well-designed or redundancy-free normalization of this schema that is also dependency-preserving does not exist.

Consequently, if one needs to guarantee the integrity of the database, and uses a dependency-preserving normal form, some redundancy must be tolerated. A natural question is then whether 3NF is the right choice of a dependency-preserving normal form. More precisely, if we look at all normal forms that guarantee dependency preservation (which excludes BCNF), and apply the information-theoretic approach to measure the amount of redundancy they introduce, will 3NF be the one with the least amount of redundancy?

Our goal in this chapter is to compute the minimum price, in terms of redundancy, that is needed to guarantee dependency preservation and show that a “good” 3NF normalization achieves this minimum redundancy. It has long been known [LTK81, Zan82] that for some schemas already in 3NF, better 3NF designs can be produced by the standard synthesis algorithm. Hence, an arbitrary 3NF schema may have quite a bit of extra redundancy. In fact, a first look [Kol05] at 3NF from the information-theoretic point of view had a rather discouraging result: for every $\varepsilon > 0$, one can find a 3NF schema with a set $\Sigma$ of FDs, an instance of that schema, and a position $p$ such that $\text{Ric}_I(p | \Sigma) < \varepsilon$. Nonetheless, the example [Kol05] showing this behavior requires arbitrarily large sets of attributes and schemas that can be further decomposed into better 3NF designs.

This gives rise to the following question: what can be said about arbitrary 3NF schemas, not only the good ones that ensure the minimum price of dependency preservation? Can they be as bad as arbitrary schemas? How do they compare to “good” 3NF
designs?

To answer these questions, we compare dependency-preserving normal forms based on the maximum redundancy that they tolerate in the instances of schemas that satisfy those normal forms. Our second main result formally confirms that some 3NF schemas may have more redundancy than others, but it also shows that arbitrary 3NF schemas have at least twice the information content compared to unnormalized schemas.

While doing a good normalization guarantees the least amount of redundancy, it may shred the original relation into too many relations, and this may affect the performance of query answering by requiring too many joins. Therefore, there is always a tradeoff between having a less redundant database and the speed of query answering. In traditional normalization theory, there is a yes or no answer to the question of whether a schema is good in terms of allowing redundant data. As the last result of this chapter, we show that there is a spectrum of redundancy ranging from too redundant to well-designed. In fact, we can evaluate the quality of a given relation schema with functional dependencies, by calculating the minimum information content, or the maximum redundancy, for instances of that schema. This number could help a database designer decide whether further normalization is necessary in case the instances of the database has the potential to carry too much redundancy.

The rest of this chapter is organized as follows. In Section 4.2, we introduce the notion of price for dependency-preserving normal forms, using the information content measure, to show that we can pay the smallest price for dependency preservation by doing a good 3NF normalization. Then in Section 4.3 we show how dependency-preserving normal forms can differ significantly in terms of the minimum information content they guarantee for each datum in a database. Furthermore, we extend our results by comparing normal forms based on the average information content they guarantee for the entire database. In Section 4.4, we show how to measure the quality of an arbitrary relation schema, in terms or redundancy, by calculating the lowest amount of information content allowed.
4.2 Price of Dependency Preservation

We say that a normal form \( \mathcal{NF} \) is dependency-preserving if every relation schema admits a dependency-preserving \( \mathcal{NF} \)-decomposition. That is, for every relation schema \((R, \Sigma)\), where \( \Sigma \) is a set of FDs, there is a lossless decomposition of \((R, \Sigma)\) into schemas \((R_1, \Sigma_1), \ldots, (R_\ell, \Sigma_\ell)\), \( \ell \geq 1 \), such that each \((R_i, \Sigma_i)\) satisfies \( \mathcal{NF} \) and \( \bigcup_{i=1}^{\ell} \Sigma_i^+ = \Sigma^+ \).

For a dependency-preserving normal form \( \mathcal{NF} \), we look at the set \( G(\mathcal{NF}) \) of values \( c \in [0, 1] \) such that for an arbitrary schema we can always guarantee an \( \mathcal{NF} \)-decomposition in which the information content in all positions is at least \( c \). We want to guarantee a certain amount of information content even for the most redundant instances of a schema. We therefore need to consider arbitrarily large instances, and hence we assume that the domain of all attributes is an infinite set, e.g., the set of positive integers \( \mathbb{N}^+ \). Formally, \( G(\mathcal{NF}) \) is the set

\[
\{ c \in [0, 1] \mid \forall (R, \Sigma), \forall I \in \text{inst}(R, \Sigma), \\
\exists \mathcal{NF}\text{-decomposition } \{(R_j, \Sigma_j)\}_{j=1}^\ell \text{ s.t. } \\
\forall j \leq \ell \forall p \in \text{Pos}(I_j), \text{ Ric}_{I_j}(p | \Sigma_j) \geq c, \}
\]

where \( I_j \) refers to \( \pi_{\text{sort}(R_j)}(I) \). Using this, we define the price of dependency preservation for \( \mathcal{NF} \) as the smallest amount of information content that is necessarily lost due to redundancies: that is, the smallest amount of redundancy one has to tolerate in order to have dependency preservation.

**Definition 4.1** For every dependency-preserving normal form \( \mathcal{NF} \), the price of dependency preservation \( \text{Price}(\mathcal{NF}) \) is defined as \( 1 - \sup G(\mathcal{NF}) \).

Clearly, \( \text{Price}(\mathcal{NF}) \leq 1 \). Since the FD-based normal form that achieves the maximum value 1 of \( \text{Ric}_I(p | \Sigma) \) in all relations is BCNF [AL05], and BCNF does not ensure dependency preservation, \( \text{Price}(\mathcal{NF}) > 0 \) for any dependency-preserving normal form \( \mathcal{NF} \).
Now we are ready to present the main result of this chapter. Intuitively, it shows that each normal form needs to pay at least half of the maximum redundancy to achieve dependency preservation, and this is exactly what 3NF pays.

**Theorem 4.2** $\text{Price}(3\text{NF}) = 1/2$. Furthermore, if $\mathcal{NF}$ is a dependency-preserving normal form, then $\text{Price}(\mathcal{NF}) \geq 1/2$.

In the rest of this section, we prove this theorem using the notations and definitions introduced in Chapter 3. We first need to define some terminology.

### 4.2.1 Guaranteed Information Content

To be able to compare normal forms with respect to the lowest amount of information content that they allow, we define a measure called guaranteed information content for a condition $\mathcal{C}$ as the smallest number $g \in [0, 1]$ that can be found as the information content of some position in instances that satisfy $\mathcal{C}$. We are sometimes interested in the lowest information content of instances of $m$-attribute relations that satisfy $\mathcal{C}$.

**Definition 4.3** Let $\mathcal{C}$ be a condition on relation schemas with functional dependencies. We define the set of possible values of $\text{Ric}_I(p \mid \Sigma)$ for $m$-attribute instances $I$ of schemas satisfying $\mathcal{C}$:

$$\mathcal{POSS}_C(m) = \{ \text{Ric}_I(p \mid \Sigma) \mid I \text{ is an instance of } (R, \Sigma), \]

$R$ has $m$ attributes,

$(R, \Sigma)$ satisfies $\mathcal{C}$.}

Then guaranteed information content, $GIC_C(m)$, is $\inf \mathcal{POSS}_C(m)$.

For instance, we know that BCNF corresponds exactly to maximum information content for all positions in all instances [AL05]. We can now formulate this fact as $GIC_{\text{BCNF}}(m) = 1$, for all $m > 0$. 


We say that a schema \((R, \Sigma)\) is **indecomposable** if it has no lossless dependency-preserving decomposition. We are only interested in indecomposable schemas that are not in BCNF since BCNF already guarantees zero redundancy. The proof of Theorem 4.2 relies on two properties of indecomposable schemas presented in propositions below. We say that a key \(X\) is **elementary** [Zan82] if there is an attribute \(A \notin X\) such that \(X' \rightarrow A \notin \Sigma^+\) for all \(X' \subseteq X\).

**Proposition 4.4** Let \(R\) be a relation schema with attributes \(\{A_1, \ldots, A_m\}\), and let \(\Sigma\) be a non-empty set of FDs over \(R\). Then \((R, \Sigma)\) is indecomposable if and only if it has an \((m - 1)\)-attribute elementary key.

**Proof:** If \((R, \Sigma)\) contains an \((m - 1)\)-attribute elementary candidate key, then every decomposition of it would lose this key; hence, it is indecomposable. Conversely, suppose \((R, \Sigma)\) is indecomposable, and there is no elementary candidate key with \(m - 1\) attributes. Let \(\Sigma_c\) be an arbitrary minimal cover for \(\Sigma\). Then for every FD \(X \rightarrow A \in \Sigma_c\), we have \(X \cup \{A\} \subseteq \text{sort}(R)\). Hence, the standard 3NF synthesis algorithm shown in Figure 4.1 will produce a dependency-preserving decomposition of \((R, \Sigma)\), and this contradicts the assumption that \((R, \Sigma)\) is indecomposable. \(\square\)

Notice that the schemas produced by the synthesis algorithm in Figure 4.1 are indecomposable. The only difference between this algorithm and the 3NF synthesis algorithm originally proposed by Bernstein [Ber76] (see Section 2.3.1) is that this one does not group the functional dependencies according to their left-hand sides before synthesizing, and may therefore produce more relations.

Let \(\mathcal{ID}\) denote the property of being indecomposable. Recall that \(\text{GIC}_{\mathcal{ID}}(m)\) is the infimum of the set \(\mathcal{POSS}_{\mathcal{ID}}(m)\) of possible values of \(\text{RIC}_{\mathcal{ID}}(p \mid \Sigma)\) for \(m\)-attribute instances of indecomposable schemas \((R, \Sigma)\). The following proposition shows that the information content in instances of indecomposable schemas can go arbitrarily close to \(1/2\) but not less than that.
**Input:** A relation schema \((R[U], \Sigma)\), where \(\Sigma\) is a set of functional dependencies.

**Output:** A database schema \(S\) in 3NF.

find a minimal cover \(\Sigma'\) of \(\Sigma\);

for each \(X \rightarrow A\) in \(\Sigma'\) do

include the relation schema \((R_i[XA], \{X \rightarrow A\})\) in the output schema:

\[
S := S \cup (R_i[XA], \{X \rightarrow A\}),
\]

where \(R_i\) is a fresh relation name;

if there is no \((R_i[U_i], \Sigma'_i)\) such that \(U_i\) is a superkey for \(R[U]\) then

choose a key \(X\) of \(R[U]\);

\[
S := S \cup \{(R_j[X], \emptyset)\},
\]

return \(S\).

**Figure 4.1:** An algorithm for synthesizing 3NF schemas [AHV95].

**Proposition 4.5** \(\text{GIC}_{DD}(m) = 1/2\) for all \(m > 2\).

Before we prove this proposition, we need a lemma. Let \(\Sigma\) be a set of FDs over a relation schema \(R\), \(I \in \text{inst}(R, \Sigma)\), \(p \in \text{Pos}(I)\). We say that \(\bar{a} \in \Omega(I, p)\) determines \(p\) if there exists \(k_0 > 0\) such that for every \(k > k_0\), we have \(P(a \mid \bar{a}) = 1\) for some \(a \in \text{adom}(I)\), and \(P(b \mid \bar{a}) = 0\) for every \(b \in [1, k] - \{a\}\). In other words, \(\bar{a}\) determines \(p\) if one can specify a single value for \(p\), given the values present in \(\bar{a}\) and constraints \(\Sigma\). We write \(\Omega_0(I, p)\) for the set of all \(\bar{a} \in \Omega(I, p)\) that determine \(p\), and \(\Omega_1(I, p)\) for the set of all \(\bar{a} \in \Omega(I, p)\) that do not determine \(p\). Let \(n = |\text{Pos}(I)|\). Then:

**Lemma 4.6** \(Ric_I(p \mid \Sigma) = |\Omega_1(I, p)|/2^{n-1}\).

**Proof of Lemma 4.6.** We show that the value of

\[
\lim_{k \to \infty} \frac{1}{\log k} \sum_{a \in [1, k]} P_k(a \mid \bar{a}) \log \frac{1}{P_k(a \mid \bar{a})}
\]

is 0 if \(\bar{a} \in \Omega_0(I, p)\) and it is 1 if \(\bar{a} \in \Omega_1(I, p)\). Assume that \(\bar{a}\) determines \(p\). By definition, there is a \(k_0 > 0\) such that for every \(k > k_0\), it is the case that \(P_k(a \mid \bar{a}) = 1\) for some
a \in \text{adom}(I)$, and $P_k(b \mid \bar{a}) = 0$ for all $b \in [1, k] - \{a\}$. Hence for all $k > k_0$ we have:

$$\sum_{a \in [1,k]} P_k(a \mid \bar{a}) \log \frac{1}{P_k(a \mid \bar{a})} = 0.$$ 

Note that $P_k(a \mid \bar{a}) \log \frac{1}{P_k(a \mid \bar{a})} = 0$ when $P_k(a \mid \bar{a}) = 0$, by definition. Then

$$\lim_{k \to \infty} \frac{1}{\log k} \sum_{a \in [1,k]} P_k(a \mid \bar{a}) \log \frac{1}{P_k(a \mid \bar{a})} = 0.$$ 

Conversely, suppose $\bar{a}$ does not determine $p$. Then for every $k_0$ there is $k > k_0$ such that either $P_k(a \mid \bar{a}) = 0$ for all $a$, or $P_k(a_1 \mid \bar{a}), P_k(a_2 \mid \bar{a}) > 0$ for at least two different values $a_1$ and $a_2$. Since $I \models \Sigma$, we have $|\text{SAT}^k_{\Sigma}(I_{(a,\bar{a})})| > 0$ for at least one $a \in \text{adom}(I)$, ruling out the first possibility. Since $\Sigma$ contains only FDs, we conclude that $|\text{SAT}^k_{\Sigma}(I_{(b,\bar{a})})| = |\text{SAT}^k_{\Sigma}(I_{(b',\bar{a})})| > 0$ for all $b, b' \not\in \text{adom}(I)$. Hence $P_k(b \mid \bar{a}) \leq 1/(k - n)$.

Next, expand $\bar{a}$ to $\bar{a}'$ by putting in a value for every position that is determined by $\bar{a}$ (which excludes $p$). Let $r$ be the number of variables in $\bar{a}'$. Then for each $c \in [1, k]$ we have $|\text{SAT}^k_{\Sigma}(I_{(c,\bar{a})})| \leq k^r$. Furthermore, for each $b \not\in \text{adom}(I)$, any substitution $\sigma$ that assigns to the $r$ variables different values in $[1, k] - (\text{adom}(I) \cup \{b\})$ will be in $\text{SAT}^k_{\Sigma}(I_{(b,\bar{a})})$; hence, we have $|\text{SAT}^k_{\Sigma}(I_{(b,\bar{a})})| \geq (k - n - r)^r$. We thus have

$$P_k(b \mid \bar{a}) \geq \frac{(k - n - r)^r}{k \cdot k^r} = \frac{1}{k} \left(1 - \frac{n + r}{k}\right)^r.$$ 

Let $\pi_i = P_k(a_i \mid \bar{a})$ for each $a_i \in \text{adom}(I)$. Then

$$\frac{1}{\log k} \sum_{a \in [1,k]} P_k(a \mid \bar{a}) \log \frac{1}{P_k(a \mid \bar{a})} \geq \frac{1}{\log k} \left(\sum_{a_i \in \text{adom}(I)} \pi_i \log \frac{1}{\pi_i} + (k - n) \cdot \frac{\log(k-n)}{k} \cdot \left(1 - \frac{n + r}{k}\right)^r\right).$$ 

Since $n$ and $r$ are fixed, this implies that $\lim_{k \to \infty} \frac{1}{\log k} \sum_{a \in [1,k]} P_k(a \mid \bar{a}) \log \frac{1}{P_k(a \mid \bar{a})} \neq 0$. By a known result [AL05], this limit always exists, and if it is not 0, then it must be equal to 1. Now we can conclude the proof of Lemma 4.6:

$$\text{RIC}_{I}(p \mid \Sigma) = \lim_{k \to \infty} \frac{1}{\log k} \sum_{\bar{a} \in \Omega_1(I,p)} \frac{1}{2^{n-1}} \sum_{a \in [1,k]} P(a \mid \bar{a}) \log \frac{1}{P(a \mid \bar{a})}$$

$$= \frac{1}{2^{n-1}} \sum_{\bar{a} \in \Omega_1(I,p)} \lim_{k \to \infty} \frac{1}{\log k} \sum_{a \in [1,k]} P(a \mid \bar{a}) \log \frac{1}{P(a \mid \bar{a})}$$

$$= |\Omega_1(I,p)|/2^{n-1}. $$
We now come back to the proof of Proposition 4.5. It consists of two parts. We prove that:

(a) for every $m > 2$ and $\varepsilon > 0$, there exists a schema $(R, \Sigma)$, an instance $I \in \text{inst}(R, \Sigma)$, and a position $p \in \text{Pos}(I)$, such that $|\text{sort}(R)| = m$, $(R, \Sigma)$ is indecomposable, and $\text{Ric}_I(p | \Sigma) < 1/2 + \varepsilon$;

(b) for every indecomposable schema $(R, \Sigma)$, every instance $I \in \text{inst}(R, \Sigma)$, and every position $p \in \text{Pos}(I)$, we have $\text{Ric}_I(p | \Sigma) \geq 1/2$.

(a) Consider the relation schema $R(A_1, \ldots, A_m)$ with FDs $\Sigma = \{A_1A_2 \ldots A_{m-1} \rightarrow A_m, A_m \rightarrow A_1\}$ and the instance $I$ of this schema shown in Figure 4.2. By Proposition 4.4, $(R, \Sigma)$ is indecomposable. Let $t_0$ denote the first tuple depicted in Figure 4.2, and let $p$ denote the position of the gray cell.

**Claim 4.7** The information content of position $p$ is

$$\text{Ric}_I(p | \Sigma) = \frac{1}{2} + \frac{1}{2} \left(\frac{3}{4}\right)^{k-1}.$$

**Proof of Claim 4.7.** Let $\bar{a}$ be an arbitrary vector in $\Omega(I, p)$. Let $\bar{a}_{[t_0]}$ denote the subtuple in $\bar{a}$ corresponding to tuple $t_0 \in I$ and $\bar{a}_{[t_1]}$ denote the subtuple in $\bar{a}$ corresponding to an arbitrary tuple $t_1 \in I$. Each position in these subtuples contains either a variable
(representing a missing value) or a constant, which equals the value that $I$ has for that position.

Then $\bar{a}$ does not determine $p$ if and only if

1. the subtuple $\bar{a}_{[t_0]}$ has a variable in the position corresponding to attribute $A_m$; or

2. the subtuple $\bar{a}_{[t_0]}$ has a constant in the position corresponding to attribute $A_m$, and for an arbitrary subtuple $\bar{a}_{[t_1]}$ in $\bar{a}$, $t_1 \neq t_0$:

   2.1. the subtuple $\bar{a}_{[t_1]}$ has a variable in the position corresponding to attribute $A_m$; or

   2.2. the subtuple $\bar{a}_{[t_0]}$ has a constant in the position corresponding to attribute $A_m$ but a variable in the position corresponding to attribute $A_1$.

In Case 1, $\bar{a}$ can have either a variable or a constant in all other $n-2$ positions. Therefore, we can have $2^{n-2}$ such $\bar{a}$’s. In Case 2, $\bar{a}_{[t_0]}$ can have either a constant or a variable in the positions corresponding to $A_2, \ldots, A_{m-1}$. Furthermore, in Case 2.1, every such subtuple $\bar{a}_{[t_1]}$ can have either a constant or a variable in the positions corresponding to attributes $A_1, \ldots, A_{m-1}$, and in Case 2.2, it can have either a constant or a variable in the positions corresponding to $A_2, \ldots, A_{m-1}$. Therefore, the total number of $\bar{a}$’s satisfying conditions of Case 2 is $2^{m-2}(2^{m-1}+2^{m-2})^{k-1}$ since we have $k-1$ tuples other than $t_0$ in the instance.

Then $|\Omega_1(I, p)|$, or the total number of different $\bar{a}$’s in $\Omega(I, p)$ that do not determine $p$ is

$$2^{n-2} + 2^{m-2}(2^{m-1}+2^{m-2})^{k-1}.$$ 

By Lemma 4.6, $\text{Ric}_I(p \mid \Sigma)$ can be obtained by dividing this number by $2^{n-1} = 2^{mk-1}$:

$$\frac{2^{mk-2} + 2^{m-2}(2^{m-1}+2^{m-2})^{k-1}}{2^{mk-1}} = \frac{1}{2} + \frac{1}{2} \left( \frac{3}{4} \right)^{k-1},$$

which proves the claim.
Thus for any $\varepsilon > 0$, there is an instance of the form shown in Figure 4.2 and a position $p$ in it such that the information content of $p$ is less than $1/2 + \varepsilon$: one needs to choose $k > 1 + \log_{4/3}(1/(2\varepsilon))$ and apply Claim 4.7.

(b) We need an easy observation (that will also be used in the proofs of the next section). For a key $X$, an attribute $A \notin X$ such that $A$ does not occur in the right-hand side of any nontrivial FD, we have $\text{Ric}_I(p \mid \Sigma) = 1$ for any instance $I$ of $(R, \Sigma)$ and any position $p$ corresponding to attribute $A$. Indeed, in this case $|\text{SAT}_{\Sigma}^k(I(a, \bar{a}))| = |\text{SAT}_{\Sigma}^k(I(b, \bar{a}))|$ for arbitrary $a, b \in [1, k]$ and hence $P(a \mid \bar{a}) = 1/k$, and thus $\text{Ric}_I^k(p \mid \Sigma) = \log k$, and $\text{Ric}_I(p \mid \Sigma) = \lim_{k \to \infty} \text{Ric}_I^k(p \mid \Sigma)/\log k = 1$.

Now let $\Sigma$ be an arbitrary non-empty set of FDs over $R(A_1, \ldots, A_m)$ such that $(R, \Sigma)$ is indecomposable, and $A_1, \ldots, A_{m-1} \rightarrow A_m \in \Sigma$ be the FD of the form described in Proposition 4.4: that is, $A_1 \ldots A_{m-1}$ is an elementary candidate key. For any instance $I$ of $(R, \Sigma)$ and any position $p = (R, t, A_m) \in \text{Pos}(I)$ corresponding to attribute $A_m$, we have $\text{Ric}_I(p \mid \Sigma) = 1$ since $p$ cannot have any redundancy due to a non-key FD.

Let $I \in \text{inst}(R, \Sigma)$, $p = (R, t_0, A_i) \in \text{Pos}(I)$, for some $i \in [1, m-1]$, and $\bar{a} \in \Omega(I, p)$. Let $\bar{a}_{[t_0]}$ denote the subtuple of $\bar{a}$ corresponding to $t_0$. It is easy to see that if $\bar{a}_{[t_0]}$ has a variable in the position corresponding to attribute $A_m$, then $\bar{a}$ does not determine $p$, no matter what the other positions in $\bar{a}$ contain. This is because there is no nontrivial FD $X \rightarrow A_i \in \Sigma^+$ such that $X \subseteq \{A_2, \ldots, A_{m-1}\}$. All other $n-2$ positions in $\bar{a}$ can therefore contain either a constant or a variable, so there are at least $2^{n-2} \bar{a}$’s that do not determine $p$. Then using Lemma 4.6, we conclude that the information content of $p$ is at least $\frac{2^{n-2}}{2^{n-1}} = 1/2$. This proves Proposition 4.5.

Now we go back to prove Theorem 4.2. The first part of the proof follows from Proposition 4.5: the information content of a position in an indecomposable instance can be arbitrarily close to $1/2$. Therefore, for every dependency-preserving normal form $\mathcal{NF}$ (which cannot further decompose an indecomposable instance), $\text{sup } G(\mathcal{NF})$ cannot exceed $1/2$. Therefore, $\text{Price}(\mathcal{NF}) \geq 1/2$. 
To prove the second part, we notice that, by Proposition 4.4 and basic properties of 3NF, every indecomposable \((R, \Sigma)\) is in 3NF. Furthermore, if \((R, \Sigma)\) is decomposable, then the 3NF synthesis algorithm of Figure 4.1 will decompose \((R, \Sigma)\) into indecomposable schemas. Therefore, for every \((R, \Sigma)\) and every \(I \in \text{inst}(R, \Sigma)\), one can find a 3NF-decomposition in which the information content of every position is at least \(1/2\) and sometimes exactly \(1/2\). That is, \(\sup \mathcal{G}(3NF) = 1/2\), and \(\text{Price}(3NF) = 1/2\). This concludes the proof.

Notice that the proof of Theorem 4.2 implies that the guaranteed information content \(1/2\) (which witnesses \(\text{Price}(3NF) = 1/2\)) occurs in decompositions produced by the standard synthesis algorithm, shown in Figure 4.1, that generates a 3NF design from a minimal cover for \(\Sigma\). Hence, our result not only justifies 3NF as the best dependency-preserving normal form, but also shows which 3NF decomposition algorithm guarantees the highest information content.

### 4.3 Comparing Normal Forms

In Section 4.2, we calculated the price of dependency preservation for normal forms and proved that one can always guarantee a 3NF decomposition whose price would be less than or equal to the price of other normal form decompositions. This good price can be achieved for schemas produced by the standard 3NF synthesis algorithm, as shown in Figure 4.1. However, not every 3NF normalization would be of the same quality in terms of redundancy. It was noticed long ago that 3NF normalization algorithms can differ significantly [LTK81, Zan82] in other aspects, such as the size of schemas they produce. In this section, we use the information-theoretic framework to compare different dependency-preserving normal forms in terms of the amount of redundancy they allow.

The measure for this comparison is the \textit{gain of normalization} function defined as

\[
\text{Gain}_{NF_1/NF_2}(m) = \frac{\text{GIC}_{NF_1}(m)}{\text{GIC}_{NF_2}(m)},
\]
where \( \text{GIC}_{\mathcal{NF}_1}(m), \text{GIC}_{\mathcal{NF}_2}(m) \) are the smallest value of \( \text{Ric}_I(p \mid \Sigma) \), as \((R, \Sigma)\) ranges over schemas with \( m \) attributes satisfying normal forms \( \mathcal{NF}_1 \) and \( \mathcal{NF}_1 \) respectively (see Definition 4.3).

We will substitute parameters \( \mathcal{NF}_1 \) or \( \mathcal{NF}_2 \) with \( \text{All} \) representing all schemas with no particular constraint, 3NF representing all schemas that satisfy the general definition of third normal form, and 3NF\(^+\) representing the indecomposable schemas generated by the synthesis algorithm shown in Figure 4.1.

We now prove that any 3NF schema, not necessarily indecomposable, is at least twice as good as some unnormalized schema. More precisely, the gain function for 3NF is the constant 2 for all \( m > 2 \) (the case of \( m \leq 2 \) is special, as any nontrivial FD over two attributes is a key, and hence all schemas are in BCNF). We also show that 3NF\(^+\) schemas could be significantly better than arbitrary 3NF schemas. That is,

**Theorem 4.8** For every \( m > 2 \):

- \( \text{Gain}_{3NF/\text{All}}(m) = 2 \);
- \( \text{Gain}_{3NF^+/3NF}(m) = 2^{m-3} \);
- \( \text{Gain}_{3NF^+/\text{All}}(m) = 2^{m-2} \).

In the proof of Theorem 4.2, we showed that \( \text{GIC}_{3NF^+}(m) = \text{GIC}_{2D}(m) = 1/2 \). Hence, the result will follow from these two propositions.

**Proposition 4.9** \( \text{GIC}_{\text{All}}(m) = 2^{1-m} \) for all \( m > 2 \).

**Proposition 4.10** \( \text{GIC}_{3NF}(m) = 2^{2-m} \) for all \( m > 2 \).

We now prove Proposition 4.9. We need to show that:

(a) for every \( m > 2 \) and \( \varepsilon > 0 \), there exists a schema \((R, \Sigma)\) with \( |\text{sort}(R)| = m \), an instance \( I \in \text{inst}(R, \Sigma) \), and a position \( p \in \text{Pos}(I) \) such that \( \text{Ric}_I(p \mid \Sigma) < 2^{1-m} + \varepsilon \);
(b) for every \((R, \Sigma)\) with \(|\text{sort}(R)| = m\), every instance \(I \in \text{inst}(R, \Sigma)\), and every position \(p \in \text{Pos}(I)\), we have \(\text{Ric}_I(p \mid \Sigma) \geq 2^{1-m}\).

(a) Consider \(R(A_1, \ldots, A_m)\) and \(\Sigma = \{A_2 \rightarrow A_1, A_3 \rightarrow A_1, \ldots, A_m \rightarrow A_1\}\). Consider the instance \(I \in \text{inst}(R, \Sigma)\) shown in Figure 4.3. Let \(t_0\) denote the first tuple depicted in this figure, and \(p = (R, t_0, A_1)\) denote the position of the gray cell. Let \(t\) be the number of tuples minus 1, that is, \((m - 1)(k - 1)\).

Claim 4.11 The information content of position \(p\) is

\[
\text{Ric}_I(p \mid \Sigma) = \frac{1}{2^{m+t-1}} \sum_{i=0}^{m-1} \binom{m-1}{i} (1 + 2^{-i})^t.
\]
Proof of Claim 4.11. Let \( \bar{a} \) be an arbitrary vector in \( \Omega(I, p) \). Let \( \bar{a}_{[t_0]} \) denote the subtuple of \( \bar{a} \) corresponding to \( t_0 \), and suppose \( \bar{a}_{[t_0]} \) has constants in positions corresponding to \( i \) attributes, and it has variables in the positions corresponding to the remaining \( m - 1 - i \) attributes. Then \( \bar{a} \) does not determine \( p \) if and only if for any arbitrary subtuple \( \bar{a}_{[t_1]} \) of \( \bar{a} \) corresponding to a tuple \( t_1 \in I, t_1 \neq t_0 \), we have:

1. the subtuple \( \bar{a}_{[t_1]} \) has a variable in the position corresponding to \( A_1 \); or

2. the subtuple \( \bar{a}_{[t_1]} \) has a constant in the position corresponding to \( A_1 \) but variables in the positions corresponding to the same \( i \) attributes for which \( \bar{a}_{[t_0]} \) has constants.

In Case 1, \( \bar{a}_{[t_1]} \) can have either a constant or a variable in every position corresponding to the other attributes \( A_2, \ldots, A_m \), and therefore there are \( 2^{m-1} \) possibilities for such subtuples. In Case 2, \( \bar{a}_{[t_1]} \) can have either a constant or a variable in every position corresponding to the other \( m - 1 - i \) attributes, and therefore there are \( 2^{m-1-i} \) such subtuples. There are \( t \) tuples in \( I \) other than \( t_0 \), and \( i \) can range over \([0, m-1]\). Therefore, \(|\Omega_1(I, p)|\) or the total number of different \( \bar{a} \)'s in \( \Omega(I, p) \) that do not determine \( p \) is

\[
\sum_{i=0}^{m-1} \binom{m-1}{i} (2^{m-1} + 2^{m-1-i})^t.
\]

The information content of \( p \) is then obtained by dividing this number by \( 2^{n-1} = 2^{m(t+1)-1} \), which proves Claim 4.11:

\[
\text{Ric}_I(p \mid \Sigma) = \frac{1}{2^{m(t+1)-1}} \sum_{i=0}^{m-1} \binom{m-1}{i} (2^{m-1} + 2^{m-1-i})^t = \frac{1}{2^{m+t-1}} \sum_{i=0}^{m-1} \binom{m-1}{i} (1 + 2^{-i})^t.
\]

The following shows that as long as \( t > \log_{4/3}(1/\varepsilon) \) (that is, \( k > (1+\log_{4/3}(1/\varepsilon))/(m-1) \)), for the instance in Figure 4.3 and position \( p \) of the gray cell the information content...
is less than $2^{1-m} + \varepsilon$:

$$
\text{Ric}_I(p \mid \Sigma) = \frac{1}{2^{m+t-1}} \sum_{i=0}^{m-1} \binom{m-1}{i} (1 + 2^{-i})^t
$$

$$
= \frac{1}{2^{m+t-1}} \left( 2^t + \sum_{i=1}^{m-1} \binom{m-1}{i} (1 + 2^{-i})^t \right)
$$

$$
< 2^{1-m} + \frac{1}{2^{m+t-1}} \sum_{i=0}^{m-1} \binom{m-1}{i} (1 + 2^{-1})^t
$$

$$
= 2^{1-m} + \frac{1}{2^{m+t-1}} \left( 2^{m-1} \left( \frac{3}{2} \right)^t \right)
$$

$$
= 2^{1-m} + \left( \frac{3}{4} \right)^t
$$

$$
< 2^{1-m} + \varepsilon.
$$

(b) Let $\Sigma$ be an arbitrary set of FDs over a relational schema $R$, $I \in \text{inst}(R, \Sigma)$, $p = (R, t_0, A_1) \in \text{Pos}(I)$, and $\bar{a} \in \Omega(I, p)$. Let $\bar{a}_{[t_0]}$ denote the subtuple in $\bar{a}$ corresponding to $t_0$. It is easy to see that if $\bar{a}_{[t_0]}$ has variables in all positions corresponding to attributes $A_2, \ldots, A_m$, then $\bar{a}$ does not determine $p$, no matter what the other positions in $\bar{a}$ contain. All the other $n - m$ positions in $\bar{a}$ can therefore contain either a constant or a variable, so the number of $\bar{a}$'s that do not determine $p$ is at least $2^{n-m}$; that is, $|\Omega_1(I, p)| \geq 2^{n-m}$. Thus, using Lemma 4.6, the information content of $p$ is at least $\frac{2^{n-m}}{2^{n-1}} = 2^{1-m}$. This proves Proposition 4.9.

Next, we prove Proposition 4.10. We need to show that:

(a) for an arbitrary $\varepsilon > 0$ and every $m > 2$, there exists a 3NF schema $(R, \Sigma)$ with $|\text{sort}(R)| = m$, an instance $I \in \text{inst}(R, \Sigma)$, and a position $p \in \text{Pos}(I)$ such that $\text{Ric}_I(p \mid \Sigma) < 2^{2-m} + \varepsilon$.

(b) for every $(R, \Sigma)$ in 3NF with $|\text{sort}(R)| = m$, every instance $I \in \text{inst}(R, \Sigma)$, and every position $p \in \text{Pos}(I)$, we have $\text{Ric}_I(p \mid \Sigma) \geq 2^{2-m}$.

(a) Consider $R(A_1, \ldots, A_m)$ and $\Sigma = \{A_1A_2 \rightarrow A_3\ldots A_m, A_3 \rightarrow A_1, \ldots, A_m \rightarrow A_1\}$. Clearly $(R, \Sigma)$ is in 3NF. Consider the instance $I \in \text{inst}(R, \Sigma)$ shown in Figure 4.2. Let
\(t_0\) denote the first tuple depicted in this figure, and \(p = (R, t_0, A_1)\) denote the position of the gray cell.

**Claim 4.12** The information content of position \(p\) is

\[
\text{RIC}_I(p \mid \Sigma) = \frac{1}{2^{m+k-3}} \sum_{i=0}^{m-2} \binom{m-2}{i} (1 + 2^{-i})^{k-1}.
\]

**Proof of Claim 4.12.** Let \(\bar{a}\) be an arbitrary vector in \(\Omega(I, p)\). Let \(\bar{a}[t_0]\) denote the subtuple in \(\bar{a}\) corresponding to \(t_0\), and suppose that \(\bar{a}[t_0]\) has constants in the positions corresponding to \(i\) attributes among \(A_3, \ldots, A_m\), and it has variables in the positions corresponding to the remaining \(m - 2 - i\) attributes. Then \(\bar{a}\) does not determine \(p\) if and only if for any arbitrary subtuple \(\bar{a}[t_1]\) in \(\bar{a}\) corresponding to a tuple \(t_1 \in I, t_1 \neq t_0\), either

1. the subtuple \(\bar{a}[t_1]\) has a variable in the position corresponding to \(A_1\); or
2. the subtuple \(\bar{a}[t_1]\) has a constant in the position corresponding to \(A_1\) but variables in the positions corresponding to the same \(i\) attributes for which \(\bar{a}[t_0]\) has constants.

In Case 1, \(\bar{a}[t_1]\) can have either a constant or a variable in every position corresponding to attributes \(A_2, \ldots, A_m\), and hence there could be \(2^{m-1}\) such subtuples for every \(t_1 \neq t_0\).

In Case 2, \(\bar{a}[t_1]\) can have either a constant or a variable in every position corresponding to the \(m - 1 - i\) attributes, and therefore there are \(2^{m-1-i}\) possible such subtuples. There are \(k - 1\) subtuples like \(\bar{a}[t_1]\), and \(i\) can range over \([0, m - 2]\). So far we have not said anything about values corresponding to \(A_2\) in \(t_0\), but since \(A_1, A_2\) is a candidate key, in both cases, \(\bar{a}[t_0]\) can have either a constant or a variable in that position. Putting it all together, we see that \(|\Omega_1(I, p)|\), the total number of different \(\bar{a}\)’s in \(\Omega(I, p)\) that do not determine \(p\) is

\[
2 \cdot \sum_{i=0}^{m-2} \binom{m-2}{i} (2^{m-1} + 2^{m-1-i})^{k-1}.
\]
The information content of $p$ can be obtained by dividing this number by $2^{n-1} = 2^{mk-1}$:

\[
Ric_I(p \mid \Sigma) = \frac{1}{2^{mk-2}} \sum_{i=0}^{m-2} \binom{m-2}{i} (2^{m-1} + 2^{m-1-i})^{k-1}
\]

\[
= \frac{1}{2^{m+k-3}} \sum_{i=0}^{m-2} \binom{m-2}{i} (1 + 2^{-i})^{k-1}.
\]

This proves Claim 4.12.

Now we need to show that for any $\varepsilon > 0$ there is an instance of the form shown in Figure 4.2 and a position $p$ in it corresponding to the gray cell such that the information content of $p$ is less than $2^{2-m} + \varepsilon$. Taking $p$ to be the position used in Claim 4.12 we have

\[
Ric_I(p \mid \Sigma) = \frac{1}{2^{m+k-3}} \sum_{i=0}^{m-2} \binom{m-2}{i} (1 + 2^{-i})^{k-1}
\]

\[
< 2^{2-m} + \frac{1}{2^{m+k-3}} \sum_{i=0}^{m-2} \binom{m-2}{i} (1 + 2^{-1})^{k-1}
\]

\[
= 2^{2-m} + \left(\frac{3}{4}\right)^{k-1}
\]

\[
< 2^{2-m} + \varepsilon,
\]

as long as $k > 1 + \log_{4/3}(1/\varepsilon)$.

(b) Let $(R, \Sigma)$ be in 3NF, $I \in inst(R, \Sigma)$, $p = (R, t_0, A_1) \in Pos(I)$, and $\bar{a} \in \Omega(I, p)$. Let $\bar{a}_{[t_0]}$ denote the subtuple in $\bar{a}$ corresponding to $t_0$. We assume that $A_1$ is a prime attribute, but not a key itself, because otherwise $Ric_I(p \mid \Sigma) = 1$ since $p$ would not have any redundancy due to a non-key FD.

It is easy to see that if $\bar{a}_{[t_0]}$ has variables in all positions corresponding to attributes $A_2, \ldots, A_m$, then $\bar{a}$ does not determine $p$, no matter what the other positions in $\bar{a}$ contain. All the other $n - m$ positions in $\bar{a}$ can therefore contain either a constant or a variable, so there are at least $2^{n-m}$ $\bar{a}$’s that do not determine $p$. Since $A_1$ is prime and not a key by itself, there is at least another attribute $A_k$ such that $A_1, A_k$ belong to a candidate key. If $\bar{a}_{[t_0]}$ has a constant in the position corresponding to $A_k$ and variables in all
positions corresponding to the other attributes, then $a$ does not determine $p$ since the FD $A_k \rightarrow A_1 \not\in \Sigma^+$. Thus, there are at least another $2^{n-m} \bar{a}$'s that do not determine $p$. Then using Lemma 4.6, the information content of $p$ is at least $\frac{2^{n-m} + 2^{n-m}}{2^{n+1}} = 2^{2-m}$, which completes the proof of Proposition 4.10 and thus of Theorem 4.8.

Combining the results of Theorem 4.8 with fact that $\text{GIC}_{\text{BCNF}}(m) = 1$, for all $m > 0$, we obtain the following comparisons of BCNF and 3NF:

**Corollary 4.13** For every $m > 2$:

- $\text{GAIN}_{\text{BCNF}/3NF^+}(m) = 2$;
- $\text{GAIN}_{\text{BCNF}/3NF}(m) = 2^{m-2}$;
- $\text{GAIN}_{\text{BCNF}/\text{All}}(m) = 2^{m-1}$.

We have so far compared normal forms based on the amount of information content that they guarantee for each position in an instance, and concluded that a 3NF+ normalization is the best according to this measure. Now we would like to extend this result by measuring the average information content over all positions in an instance, and see if 3NF+ guarantees the best average among all dependency-preserving normal forms.

### 4.3.1 Guaranteed Average Information Content

To compare normal forms with respect to the lowest average information content that they allow for an instance, we define a measure called *guaranteed average information content* for a condition $C$ as the smallest number $g \in [0, 1]$ that can be found as the average information content of some instance that satisfies $C$. We are sometimes interested in the lowest average information content of instances of $m$-attribute relations that satisfy $C$.

**Definition 4.14** Let $C$ be a condition on relation schemas with functional dependencies. Let $\text{AVG}(I \mid \Sigma)$ denote the average of the numbers in $\{\text{RIC}_I(p \mid \Sigma) \mid p \in \text{Pos}(I)\}$. We
define the set of possible values of $\text{AVG}(I \mid \Sigma)$ for $m$-attribute instances $I$ of schemas satisfying $C$:

$$\mathcal{POSS}_C(m) = \{ \text{AVG}(I \mid \Sigma) \mid I \text{ is an instance of } (R, \Sigma),
\begin{align*}
R &\text{ has } m \text{ attributes,} \\
(R, \Sigma) &\text{satisfies } C \}.
\end{align*}$$

Then guaranteed average information content, $\text{GAVG}_C(m)$, is $\inf \mathcal{POSS}_C(m)$.

Again, we can clearly say that $\text{GAVG}_{\text{BCNF}}(m) = 1$, for all $m > 0$. We now want to show that doing a $3\text{NF}^+$ normalization can also have a significant effect on the average information content of instances. We do this by calculating a lower bound for $\text{GAVG}_{3\text{NF}^+}$ and an upper bound for $\text{GAVG}_{\text{All}}$ and $\text{GAVG}_{3\text{NF}}$, as shown by the following propositions.

**Proposition 4.15** $\text{GAVG}_{3\text{NF}^+}(m) \geq 1/2 + 1/(2m)$ for all $m > 2$.

**Proposition 4.16** $\text{GAVG}_{3\text{NF}}(m) \leq 2^{1 - \frac{m}{2}}$ for all $m > 4$.

We now prove Proposition 4.15. We need to show that for every $(R, \Sigma)$ with $|\text{sort}(R)| = m$ and every instance $I \in \text{inst}(R, \Sigma)$, we have $\text{AVG}(I \mid \Sigma) \geq 1/2 + 1/(2m)$. Let $\Sigma$ be an arbitrary non-empty set of FDs over $R(A_1, \ldots, A_m)$ such that $(R, \Sigma)$ is indecomposable, and $A_1, \ldots, A_{m-1} \rightarrow A_m \in \Sigma$ be the FD of the form described in Proposition 4.4.

For an instance $I \in \text{inst}(R, \Sigma)$ and any position $p = (R, t, A_m) \in \text{Pos}(I)$, we have $\text{RIC}_I(p \mid \Sigma) = 1$. If $k$ is the total number of tuples in $I$, then there is $k$ of these positions. For all other positions $p = (R, t, A_i), i \in [1, m-1]$ corresponding to any of the attributes $A_1, \ldots, A_{m-1}$, we have $\text{RIC}_I(p \mid \Sigma) \geq 1/2$, by Proposition 4.5, and there is $k \cdot (m - 1)$ of these positions. If $n$ denotes $\|I\|$, then

$$\text{AVG}(I \mid \Sigma) \geq \frac{k + k \cdot (m - 1) \cdot \frac{1}{2}}{n} = \frac{\frac{1}{2} + \frac{1}{2m}}{n},$$

which proves the proposition.
To prove Proposition 4.16, we show that for every $m > 4$ and $\varepsilon > 0$, there exists a schema $(R, \Sigma)$ with $|\text{sort}(R)| = m$ and an instance $I \in \text{inst}(R, \Sigma)$, such that $(R, \Sigma)$ is in 3NF, and $\text{AVG}(I \mid \Sigma) < 2^{1-m/2} + \varepsilon$. Consider $R(A_1, \ldots, A_m)$, where $m$ is an even integer and

$$
\Sigma = \{A_1 \rightarrow A_3, A_3 \rightarrow A_5, \ldots, A_{m-3} \rightarrow A_{m-1}, A_{m-1} \rightarrow A_1, A_2 \rightarrow A_4, A_4 \rightarrow A_6, \ldots, A_{m-2} \rightarrow A_m, A_m \rightarrow A_2\}.
$$

Consider the instance $I \in \text{inst}(R, \Sigma)$ shown in Figure 4.4. Let $t_0$ denote the first tuple depicted in this figure, and $p = (R, t_0, A_1)$ denote the position of the gray cell.
Claim 4.17 The information content of position $p$ is

$$\text{Ric}_1(p \mid \Sigma) = \frac{1}{2^{\frac{m}{2} + k - 2}} \sum_{i=0}^{m-1} \left( \frac{m}{2} - 1 \right) \left(1 + 2^{-i} \right)^{k-1}.$$ 

Proof of Claim 4.17. Let $\bar{a}$ be an arbitrary vector in $\Omega(I, p)$. Let $\bar{a}_{[t_0]}$ denote the subtuple in $\bar{a}$ corresponding to $t_0$, and suppose that $\bar{a}_{[t_0]}$ has constants in the positions corresponding to $i$ attributes among $A_3, A_5, \ldots, A_{m-1}$, and it has variables in the positions corresponding to the remaining $m/2 - i$ attributes with odd subscripts. Then $\bar{a}$ does not determine $p$ if and only if for any arbitrary subtuple $\bar{a}_{[t_1]}$ in $\bar{a}$ corresponding to a tuple $t_1$ among the first $k$ tuples in Figure 4.4, $t_1 \neq t_0$, either

1. the subtuple $\bar{a}_{[t_1]}$ has a variable in the position corresponding to $A_1$; or

2. the subtuple $\bar{a}_{[t_1]}$ has a constant in the position corresponding to $A_1$ but variables in the positions corresponding to the same $i$ attributes among $A_3, A_5, \ldots, A_{m-1}$ for which $\bar{a}_{[t_0]}$ has constants.

In Case 1, $\bar{a}_{[t_1]}$ can have either a constant or a variable in every position corresponding to attributes $A_2, \ldots, A_m$, and hence there could be $2^{m-1}$ such subtuples for every $t_1 \neq t_0$. In Case 2, $\bar{a}_{[t_1]}$ can have either a constant or a variable in every position corresponding to the $m - 1 - i$ attributes, and therefore there are $2^{m-1-i}$ possible such subtuples. There are $k - 1$ subtuples like $\bar{a}_{[t_1]}$, and $i$ can range over $[0, m/2 - 1]$. So far we have not said anything about positions corresponding to attributes $A_2, A_4, \ldots, A_m$ in $t_0$ ($m/2$ positions) and all positions in tuples that are not among the first $k$ tuples in Figure 4.4 ($mk(k-1)$ positions). Since these positions do not have anything to do with the value in position $p$, $\bar{a}$ can have either a constant or a variable for those positions and still do not determine $p$. Putting it all together, we see that $|\Omega_1(I, p)|$, the total number of different $\bar{a}$’s in $\Omega(I, p)$ that do not determine $p$ is

$$2^{\frac{m}{2}} \cdot 2^{mk(k-1)} \cdot \sum_{i=0}^{m-1} \left( \frac{m}{2} - 1 \right) \left(2^{m-1} + 2^{m-1-i} \right)^{k-1}.$$
The information content of $p$ can be obtained by dividing this number by $2^n - 1 = 2^{mk^2 - 1}$:

$$\text{Ric}_I(p | \Sigma) = \frac{2^m \cdot 2^{mk(k-1)} \sum_{i=0}^{\frac{m}{2} - 1} \left( \frac{m}{2} - 1 \right) (2^{m-1} + 2^{m-1-i})^{k-1}}{2^{\frac{m}{2} + k-2} \sum_{i=0}^{\frac{m}{2} - 1} \left( \frac{m}{2} - 1 \right) (1 + 2^{-i})^{k-1}}$$

This proves Claim 4.17.

Now we proceed with the proof of Proposition 4.16. It can be easily observed that instance $I \in \text{inst}(R, \Sigma)$ shown in Figure 4.4 is symmetric with respect to all positions, and therefore the information content of all positions is the same as the value that we calculated in Claim 4.17 since the FDs are also symmetric. Then we have

$$\text{AVG}(I | \Sigma) = \frac{1}{2^{\frac{m}{2} + k-2}} \sum_{i=0}^{\frac{m}{2} - 1} \left( \frac{m}{2} - 1 \right) (1 + 2^{-i})^{k-1}$$

$$< 2^{1-\frac{m}{2}} + 2^{1-\frac{m}{2}} \cdot \sum_{i=1}^{\frac{m}{2} - 1} \left( \frac{m}{2} - 1 \right) (1 + 2^{-1})^{k-1}$$

$$= 2^{1-\frac{m}{2}} + 2^{1-\frac{m}{2}} \cdot (\frac{m}{2} - 1) \cdot \left( \frac{3}{4} \right)^{k-1}$$

$$< 2^{1-\frac{m}{2}} + \varepsilon$$

if we take $k > 1 + \log_{4/3}(\frac{m/2 - 1}{2^{m/2 - 1}})$, which completes the proof of Proposition 4.16.

### 4.4 How Good Is an Arbitrary Design?

So far we have calculated the lowest information content that normal forms allow. However, schemas are often not like the extreme examples that we presented as the worst cases of normal forms. Now the question is how good an arbitrary schema is in terms of redundancy to decide whether normalization is necessary if the schema is allowing too much redundancy. This can be useful to ensure that schemas are not overnormalized, as overnormalizing databases affects the performance of query answering. In this
section, we calculate the minimum information content for an arbitrary relation schema with functional dependencies. We first extend the definition of guaranteed information content for an arbitrary schema:

**Definition 4.18** Let $R$ be a relation schema and $\Sigma$ be a set of functional dependencies defined over the attributes of $R$. For an attribute $A \in \text{sort}(R)$, we define the set of possible values of $\text{Ric}_I(p \mid \Sigma)$ for positions $p = (R, t, A)$ in instances $I \in \text{inst}(R, \Sigma)$:

$$\text{POSS}_{\Sigma}^R(A) = \{\text{Ric}_I(p \mid \Sigma) \mid I \text{ is an instance of } (R, \Sigma), \ p = (R, t, A) \text{ is in } \text{Pos}(I)\}.$$ 

Then guaranteed information content of schema $(R, \Sigma)$ for column $A$, $\text{GIC}_\Sigma^R(A)$, is $\inf \text{POSS}_{\Sigma}^R(A)$.

In other words, $\text{GIC}_\Sigma^R(A)$ is the least amount of information content that may be found in $A$-columns of instances of $R$ that satisfy FDs in $\Sigma$. Obviously, $\text{GIC}_\Sigma^R(A) = 1$ for every attribute $A$ that is not implied by any non-key set of attributes, i.e. when there is no FD $X \rightarrow A$, such that $X$ is not a superkey. For other attributes, the following theorem shows how to calculate the value of $\text{GIC}_\Sigma^R(A)$ by finding all minimal non-key sets of attributes that imply $A$ (no proper subset of them imply $A$). We will also need to find the number of *hitting sets* for a collection of subsets of attributes. Given a collection of subsets of a universe, a hitting set (or a hypergraph traversal) is a subset of the universe that intersects every set in the collection.

**Theorem 4.19** Let $I_{\min}^A$ denote the set $\{X \mid X \text{ is a minimal non-key subset of } \text{sort}(R), \ X \rightarrow A \in \Sigma^+, \ A \notin X\}$, and $\#\text{HS}(I_{\min}^A)$ be the number of all hitting sets of $I_{\min}^A$. Then

$$\text{GIC}_\Sigma^R(A) = \#\text{HS}(I_{\min}^A) \cdot 2^{-l},$$

where $l$ is the cardinality of $\bigcup_{X \in I_{\min}^A} X$. 
Proof: Let $X_1, \ldots, X_k$ denote the sets in $I^A_{\text{min}}$, and $S$ contain all the hitting sets of the collection $I^A_{\text{min}}$, i.e.,

$$S = \{Y \mid Y \subseteq X_1 \cup \ldots \cup X_k, Y \cap X_i \neq \emptyset \text{ for all } i \in [1, k]\}.$$ 

Then the proof consists of two parts. We prove that:

(a) for every $\varepsilon > 0$, there exists an instance $I \in \text{inst}(R, \Sigma)$ and a position $p = (R, t, A)$ in $\text{Pos}(I)$, such that $\text{Ric}_I(p \mid \Sigma) < |S| \cdot 2^{-l} + \varepsilon$;

(b) for every instance $I \in \text{inst}(R, \Sigma)$ and every position $p = (R, t, A)$ in $\text{Pos}(I)$, we have $\text{Ric}_I(p \mid \Sigma) \geq |S| \cdot 2^{-l}$.

(a) We construct instance $I$ of $R$ consisting of tuples $t_0, \ldots, t_q$, where $q = kr$, as follows:

- for all attributes $B \in \text{sort}(R)$, $t_0[B] = 1$;
- for a tuple $t_i, i \in [1, q], t_i[B] = 1$ for all $B \in X^+_j$, where $j = \lceil i/r \rceil$, and for all other attributes $C \notin X^+_j, t_i[C] = v$, where $v$ is a fresh value not used so far.

This instance consists of $k$ groups, each having $r$ tuples that agree with each other and also with tuple $t_0$ only on one of the sets of attributes $X^+_j$. Now consider position $p = (R, t_0, A)$.

Claim 4.20 For the information content of position $p$ we have

$$\text{Ric}_I(p \mid \Sigma) \leq |S| \cdot 2^{-l} + \frac{2^{-l} - |S|}{2^l} \cdot \left(\frac{2^m - 1}{2^m}\right)^r,$$

where $m = |\text{sort}(R)|$.

Proof of Claim 4.20. Let $\bar{a}$ be an arbitrary vector in $\Omega(I, p)$. Let $\bar{a}_{[t_0]}$ denote the subtuple in $\bar{a}$ corresponding to tuple $t_0 \in I$. If for all attributes in one of the sets $Y$ in $S$, $\bar{a}_{[t_0]}$ contains variables, then $\bar{a}$ does not determine $p$ no matter what the other positions in $\bar{a}$ contain. This is because none of the FDs implying attribute $A$ could enforce a value
for position \( p \) since \( Y \) contains at least one element from each \( X_j, j \in [1, k] \). There are 
\(|S| \cdot 2^{n-l-1}\) of such \( \bar{a} \)'s.

If for all attributes of some \( X_j, j \in [1, k] \), subtuple \( \bar{a}_{[t_0]} \) contains constants, which 
can happen for \((2^l - |S|) \cdot 2^{m-l-1}\) subtuples, then \( \bar{a} \) does not determine \( p \) only if for any 
subtuple \( \bar{a}_{[t_i]} \) corresponding to a tuple \( t_i, i \in ((j-1)r, jr) \), \( \bar{a}_{[t_i]} \) does not contain constant 
for at least one attribute in \( X_j.A \). Therefore, \( \bar{a}_{[t_i]} \) can have at most \( 2^m - 1 \) shapes. The 
other \( q-r \) subtuples in \( \bar{a} \) can have at most \( 2^m \) shapes.

Putting everything together, \(|\Omega_1(I, p)|\), or the total number of different \( \bar{a} \)'s in \( \Omega(I, p) \) 
that do not determine \( p \) is at most
\[
|S| \cdot 2^{n-l-1} + (2^l - |S|) \cdot (2^m - 1)^r \cdot (2^m)^q-r \cdot 2^{m-l-1},
\]
and \( n = m(q+1) \). Then by Lemma 4.6,
\[
\text{Ric}_I(p \mid \Sigma) \leq \frac{|S| \cdot 2^{n-l-1}}{2^{n-l-1}} + \frac{(2^l - |S|) \cdot (2^m - 1)^r \cdot (2^m)^q-r \cdot 2^{m-l-1}}{2^{n-l-1}} = |S| \cdot 2^{-l} + \frac{2^l - |S|}{2^l} \cdot \frac{(2^m - 1)}{2^m},
\]
which proves the claim. By taking \( r > \log \frac{2^m}{2^m-|S|} \) we will have \( \text{Ric}_I(p \mid \Sigma) < |S| \cdot 2^{-l} + \varepsilon \).

(b) Let \( I \) be an arbitrary instance in \( \text{inst}(R, \Sigma), p = (R, t_0, A) \in \text{Pos}(I) \), and \( \bar{a} \in \Omega(I, p) \).
Let \( \bar{a}_{[t_0]} \) denote the subtuple of \( \bar{a} \) corresponding to \( t_0 \). If for all attributes in one of the 
sets \( Y \) in \( S \), \( \bar{a}_{[t_0]} \) contains variables, then \( \bar{a} \) does not determine \( p \) no matter what the 
other positions in \( \bar{a} \) contain. There are \(|S| \cdot 2^{n-l-1}\) of such \( \bar{a} \)'s. Therefore, \(|\Omega_1(I, p)|\) is at 
least \(|S| \cdot 2^{n-l-1}\), and thus \( \text{Ric}_I(p \mid \Sigma) \geq |S| \cdot 2^{-l} \).

**Example 4.21** Consider relation schema \( R_1(A, B, C, D, E) \) and the following set of 
FDs:
\[
\Sigma_1 = \{ \ AB \rightarrow E, \quad D \rightarrow E \}.
\]
We use Theorem 4.19 to calculate the minimum information content, or the maximum redundancy, allowed in column $E$ of instances of $R_1$ satisfying $\Sigma_1$:

$$I_{\text{min}}^E = \{AB, D\}$$

$$\#\text{HS}(I_{\text{min}}^E) = |\{AD, BD, ABD\}| = 3$$

$$l = 3$$

$$\text{GIC}^{R_1}_{\Sigma_1}(E) = 3 \cdot 2^{-3} = \frac{3}{8}$$

Example 4.22 Consider relation schema $R_2(A, B, C, D, E)$ and the following set of FDs:

$$\Sigma_2 = \{A \rightarrow B, AC \rightarrow E, BD \rightarrow E \}$$

The minimum information content allowed by this schema can be calculated as follows:

$$I_{\text{min}}^E = \{AC, BD, AD\}$$

$$\#\text{HS}(I_{\text{min}}^E) = |\{AD, AB, CD, ABC, ABD, ACD, BCD, ABCD\}| = 8$$

$$l = 4$$

$$\text{GIC}^{R_2}_{\Sigma_2}(E) = 8 \cdot 2^{-4} = \frac{1}{2}$$

By looking at the two sets of FDs in Examples 4.21 and 4.22, it may not be immediately obvious that which one of the designs $(R_1, \Sigma_1)$ or $(R_2, \Sigma_2)$ allow more redundancy in column $E$. However, according to our calculations the first schema has the potential to produce more redundant instances.

4.5 Related Work

Information theory has previously been used to reason about database dependencies in several papers [Lee87, CP87, DR00, LL03]. Most of the works in this area try to quantify the information associated with a set of attributes, and use the concept of conditional
entropy in the following way. Given two sets of attributes \( X \) and \( Y \), the conditional entropy \( H(Y \mid X) \) intuitively means how much information is provided by \( Y \) knowing that the values of attributes in \( X \) are given. When the functional dependency \( X \rightarrow Y \) holds, we have \( H(Y \mid X) = 0 \), no matter how the instances look. This means that once the values for attributes in \( X \) are known, there is no uncertainty regarding the values for attributes in \( Y \).

For instance, Cavallo et al. [CP87] use the notion of entropy in the context of probabilistic databases with a similar definition. Dalkilic et al. [DR00] introduce the notion of information dependency measure between two sets of attributes \( X \) and \( Y \) that intuitively means how much do we not know about \( Y \) provided we know about \( X \). Lee [Lee87] also defines the concept of conditional entropy for two sets of attributes in a similar way. Using these definitions, one cannot measure the amount of redundancy that functional dependencies cause in an instance, and it is not possible to compare the quality of schemas in terms of the redundancy they impose.

Some old papers [LTK81, Zan82] have addressed the problem of comparing the quality of schemas that are in 3NF, and have tried to give an improved definition of this normal form. Zaniolo [Zan82] provides a simpler definition for 3NF compared to the original definition of Codd [Cod72], and shows that his definition corresponds to the 3NF schemas produced by Bernstein’s synthesis algorithm [Ber76].

Ling et al. [LTK81] give a new definition for 3NF to improve the redundancy of schemas by eliminating *superfluous* attributes: attributes whose value can always be derived from other attributes (perhaps using other relations), and whose value is not needed to derive other attributes’ values. None of the works explained above compare the redundancy of possible 3NF decompositions for a given schema, which is a major concern of this chapter.
Chapter 5

XML Documents - An Overview

In this chapter, we review basic notions of XML databases, such as XML trees and DTDs, and present some of the recent proposals for XML integrity constraints as well as the existing design principles for XML data.

5.1 DTDs and XML Documents

The eXtensible Markup Language (XML) is a simple and flexible text format that has recently become the standard language for exchanging a wide variety of data on the web. An XML document contains one or more elements that are delimited by matching start and end tags. For example, in the XML document shown in Figure 5.1, the start tag <type> and the corresponding end tag </type> specify an element that contains the text consulting: <type> consulting </type>.

Every element may contain text data, other elements, or a mixture of them. It can also contain attributes. For example, the element that starts with the tag <branch> in Figure 5.1 contains three subelements, one starting with the tag <type> and the other two starting with the tag <client>, and an attribute bid that has the value "co201".

To define a logical structure for a class of XML documents, we need to specify a schema that defines the legal building blocks of each document. There are two common ways to
do this, neither of which has yet become the standard way: Document Type Definitions (DTDs) [Gol90, Hun00] and W3C XML Schema [TBME, vdV02]. The language of XML Schema definitions is richer compared to that of DTDs: XML Schema supports data types and namespaces, while DTD does not. However, DTDs have been of more interest to the database research community, and thus we consider only DTDs since they have been used by papers that address integrity constraints and design issues for XML data.

A DTD describing the company database is shown in Figure 5.2. This DTD specifies the set of elements allowed in each document of the database, and for each element it specifies the set of subelements and attributes. For example, the text segment

\[
\text{<!ELEMENT branch type(client*)>}
\]

is an element declaration that specifies that the subelements of each element of type \textit{branch} form a string in the regular language \textit{type(client)*}. If an element structure is described as \texttt{#PCDATA}, it means that the elements of this type contain text data. The attributes for each element are specified using an \texttt{ATTLIST} declaration. A \texttt{DOCTYPE} declaration specifies the root element of the document.

Next we present a formal model for XML documents and DTDs, proposed by Fan and Libkin [FL01, FL02], and review some basic concepts such as paths in DTDs and XML documents. Assume that we have the following disjoint sets: \textit{El} of element names, \textit{Att} of attribute names, \textit{Str} of possible values of string-valued attributes, and \textit{Vert} of node identifiers. All attribute names start with the symbol @.

**Definition 5.1** An XML tree is defined to be a tree \( T = (V, \text{lab}, \text{ele}, \text{att}, \text{root}) \), where

- \( V \subseteq \text{Vert} \) is a finite set of nodes.
- \( \text{lab} : V \rightarrow \text{El} \) assigns a label to each node of the tree.
- \( \text{ele} : V \rightarrow \text{Str} \cup V^* \) assigns to each node a string or an ordered set of nodes as its children.
- \( \text{att} \) is a partial function of type \( V \times \text{Att} \rightarrow \text{Str} \). For each \( v \in V \), the set \( \{ @l \in \text{Att} \mid \text{att}(v, @l) \text{ is defined} \} \) is finite.


<company>
  <branch bid="co201">
    <type> consulting </type>
    <client name="cl1">
      <contact areaCode="416" phone="860 1212" city="Toronto">
      </contact>
    </client>
    <client name="cl2">
      <contact areaCode="613" phone="520 9191" city="Ottawa">
      </contact>
    </client>
  </branch>
</company>

Figure 5.1: An XML document.

- root ∈ V is the root node of the tree T.

We assume that there is a parent-child relation on the nodes of a tree: \{(v, v') ∈ V × V | v' occurs in ele(V)\}. For each v ∈ V, the set of all v' ∈ V that occur in ele(v) are called subelements or children of v, and the set \{ @l ∈ Att | att(v, @l) is defined \} is called attributes of node v. For each v ∈ V, lab(v) is also referred to as type of node v.

We also adopt the following notation [FL01, FL02, Are05]. Given an XML tree T and an element type τ ∈ El, ext(τ) is the set of all nodes of T that are of type τ. If v is a node of T and @l ∈ Att such that att(v, @l) is defined, then we sometimes write v.@l instead of att(v, @l). Given a set of attributes X = [@l_1, ..., @l_n] and a node v such that att(v, @l_i) is defined for every i ∈ [1, n], v[X] is defined to be the list of values [att(v, @l_1), ..., att(v, @l_n)].
Chapter 5. XML Documents - An Overview

Figure 5.2: A DTD for the company database.

Example 5.2 The tree representation of the XML document in Figure 5.1 is shown in Figure 5.3. The set of nodes is \( V = \{ v_i \mid i \in [0,12] \} \), and \( v_0 \) is the root. Parts of the functions \( \text{lab}, \text{ele}, \) and \( \text{att} \) are shown below.

\[
\begin{align*}
\text{lab}(v_0) &= \text{company} \\
\text{ele}(v_0) &= [v_1, v_7] \\
\text{lab}(v_1) &= \text{branch} \\
\text{ele}(v_1) &= [v_2, v_3, v_5] \\
v_1.@\text{bID} &= "\text{co201}\" \\
\text{lab}(v_2) &= \text{type} \\
\text{ele}(v_2) &= [\text{consulting}] \\
\text{lab}(v_3) &= \text{client} \\
\text{ele}(v_3) &= [v_4] \\
v_3.@\text{name} &= "\text{cl1}\" \\
\text{lab}(v_4) &= \text{contact} \\
\text{ele}(v_4) &= \epsilon \\
v_4.@\text{areaCode} &= "416"
\end{align*}
\]

Figure 5.3 also shows that \( \text{ext} (\text{branch}) = \{ v_1, v_7 \} \), \( \text{ext} (\text{type}) = \{ v_2, v_8 \} \), \( \text{ext} (\text{client}) = \{ v_3, v_5, v_9, v_{11} \} \), and \( \text{ext} (\text{contact}) = \{ v_4, v_6, v_{10}, v_{12} \} \).
Definition 5.3 A DTD (Document Type Definition) is defined to be \( D = (E, A, P, R, r) \), where

- \( E \subseteq El \) is a finite set of element types.
- \( A \subseteq Att \) is a finite set of attributes.
- \( P \) maps each element in \( E \) to an element type definition. For an element \( \tau \in E \),
  \[ P(\tau) = S \text{ representing } PCDATA \text{ or } P(\tau) \text{ is a regular expression } \alpha \text{ over } E - \{r\}, \]
  defined as follows:
  \[ \alpha ::= \epsilon | \tau' | \alpha \sigma | \alpha, \alpha | \alpha^*, \]
  where \( \epsilon \) represents the empty string, \( \tau' \in E \), and “|”, “,”, and “*” denote union, concatenation, and the Kleene star closure respectively.
- \( R \) assigns a subset of \( A \) to each element \( \tau \in E \). If \( @l \in R(\tau) \), we say that \( @l \) is defined for \( \tau \).
- \( r \in E \) is the root element type.

For simplicity, we sometimes do not consider PCDATA elements in XML trees since they can be represented by attributes. In that case, \( P \) is just a set of rules that map each element type to a regular expression.
Example 5.4 The DTD in Figure 5.2 can be formally described as $D = (E, A, P, R, r)$, where
\[
E = \{\text{company, branch, type, client, contact}\},
\]
\[
A = \{\text{@bID, @name, @areaCode, @phone, @city}\},
\]
\[
r = \text{company}.
\]
Furthermore, $R$ and $P$ are defined as follows:
\[
R(\text{company}) = \emptyset \quad R(\text{branch}) = \{\text{@bID}\}
\]
\[
R(\text{type}) = \emptyset \quad R(\text{client}) = \{\text{@name}\}
\]
\[
R(\text{contact}) = \{\text{@areaCode, @phone, @city}\}
\]
\[
P(\text{company}) = \text{branch}^* \quad P(\text{client}) = \text{contact}^* \quad P(\text{type}) = S
\]
\[
P(\text{branch}) = \text{type, client}^* \quad P(\text{contact}) = \epsilon.
\]

We now present the formal definition [FL01, FL02] of what it means for an XML tree to conform to a DTD.

Definition 5.5 Given a DTD $D = (E, A, P, R, r)$ and an XML tree $T = (V, \text{lab}, \text{ele}, \text{att}, \text{root})$, we say that $T$ conforms to $D$, written as $T \models D$, if

- lab is a mapping from $V$ to $E$;

- for each $v \in V$, if $P(\text{lab}(v)) = S$, then $\text{ele}(v) = [s]$ for some $s \in \text{Str}$; otherwise, $\text{ele}(v) = [v_1, \ldots, v_n]$, where the string $\text{lab}(v_1) \ldots \text{lab}(v_n)$ is in the language defined by $P(\text{lab}(v))$;

- att is a partial function from $V \times A$ to Str such that for every $v \in V$ and $@l \in A$, $\text{att}(v, @l)$ is defined if and only if $@l \in R(\text{lab}(v))$;

- lab(root) = r.
For example, the XML tree in Example 5.2 conforms to the DTD defined in Example 5.4.

The notion of \textit{path} is used to navigate and query XML trees and is also used to define constraints for XML. Given an XML tree \( T = (V, \text{lab}, \text{ele}, \text{att}, \text{root}) \), a path in \( T \) is a string \( w = w_1 \ldots w_n \), where \( w_1, \ldots, w_{n-1} \in \text{El} \) and \( w_n \in \text{El} \cup \text{Att} \cup \{S\} \), such that there are vertices \( v_1, \ldots, v_{n-1} \) in \( V \) with labels \( w_1, \ldots, w_{n-1} \) respectively, such that

- \( v_{i+1} \) is a child of \( v_i \), \( i \in [1, n-2] \),
- if \( w_n \in \text{El} \) then \( v_{n-1} \) has a child \( v_n \) with label \( w_n \). If \( w_n = @l \) is an attribute in \( \text{Att} \), then \( \text{att}(v_{n-1}, @l) \) is defined. If \( w_n = S \), then \( v_{n-1} \) has a child in \( \text{Str} \).

The set of all paths in a tree \( T \) that start from the root is denoted by \( \text{paths}(T) \). Given two nodes \( x, y \) in \( T \) such that \( y \) is a descendant of \( x \), we say that \( w_1, \ldots, w_n \) is a path from \( x \) to \( y \) if in the above definition we have \( v_1 = x \) and \( v_n = y \).

Paths can also be defined for DTDs. Given a DTD \( D = (E, A, P, R, r) \), a path in \( D \) is a string \( w = w_1, \ldots, w_n \) such that \( w_i \) is in the alphabet of \( P(w_{i-1}) \) for \( i \in [2, n-1] \), and \( w_n \) is either an attribute \( @l \in R(w_{n-1}) \) or is in the alphabet of \( P(w_{n-1}) \). The length of the path \( w \) is denoted by \( \text{length}(w) \) and the last symbol on the path is denoted by \( \text{last}(w) \). If \( \text{last}(w) \in E \), we say that \( w \) is an \textit{element path}. If \( w \) ends with an attribute, we say that it is an \textit{attribute path}. The set of all element paths defined for a DTD \( D \) is denoted by \( \text{EPath}(D) \).

\section{5.2 Key and Foreign Key Constraints for XML}

Keys or “unique identifiers” play an essential role in the design stage of a database and later in deletions and updates. They provide a way to refer to an object in a database, such as a tuple in a relational database or an element in an XML document. Foreign keys are attributes by which a tuple or object can refer to other tuples or objects in the database instance. Here we review the proposals for defining key and foreign key constrains for XML documents.
In W3C XML standard [BPSM⁺, BPSM97], one can define for each element type at most one ID attribute in the DTD. An element can be uniquely identified within a document by the value of its ID attribute. Then other elements can refer to this element by their IDREF attributes. The value of each IDREF attribute should match the value of an ID attribute of some element in the document. For example, in any XML document conforming to the following DTD, each *book* element has a unique value for its *isbn* attribute, and each *ref* element refers to an ID of some book by the value of its *to* attribute.

```xml
<!DOCTYPE publications [
  <!ELEMENT book (author+, ref*)>
  <!ATTLIST book
    isbn ID #required>
  <!ELEMENT author (#PCDATA)>
  <!ELEMENT ref EMPTY>
  <!ATTLIST ref
    to IDREF #implied>
]
```

This mechanism however resembles a local pointer system rather than a way to define key constraints: first, ID attributes determine elements within the entire document rather than among the elements of the same type; second, only one attribute of each element type can be defined as an ID and hence only one unary key is allowed for each element; and third, there is no clear scope for IDREF pointers; each IDREF attribute can point to any element type in the document.

In XML Schema Recommendations by W3C [TBME, vdV02], it is possible to define more expressive constraints, which look more like key specifications. These constraints enable us to indicate that a set of attributes or elements must be unique within a certain
scope. To do this, we define a *unique* constraint by specifying the following components. First, we *select* a set of elements and then identify the attributes or elements *field* relative to each selected element that have to be unique within the scope of the selected elements. Selector and field are specified by expressions in the W3C XPath language [CE, HM04], which is a language to address parts of an XML document. For example, in the following specification, the constraint `key1` specifies that the value of attribute `@isbn` is unique within the scope of `book` elements.

```xml
<unique name="key1">
  <selector xpath="book"/>
  <field xpath="@isbn"/>
</unique>
```

A number of more formal key specifications for XML have been proposed, and we present some of them in the rest of this section. We first introduce a key and foreign key language initially proposed by Fan and Siméon [FS00, FS03] and further studied and extended by Fan et al. [FL01, FL02, AFL02a, AFL02b]. Then we present another proposal for XML keys and foreign keys.

### 5.2.1 Keys Defined by Attributes

A class of absolute keys and foreign keys was first introduced by Fan and Siméon [FS00, FS03], which defines a key of an element in terms of a set of attributes of that element. Given a DTD \(D = (E, A, P, R, r)\), an *absolute key* is an expression of the form \(\tau[X] \rightarrow \tau\), where \(\tau \in E\), and \(X\) is a nonempty set of attributes in \(R(\tau)\). Let \(T\) be an XML tree that conforms to \(D\) and \(\text{ext}(\tau)\) denote the set of all nodes in \(T\) labeled \(\tau\). Then \(T\) satisfies the key constraint \(\tau[X] \rightarrow \tau\), written as \(T \models \tau[X] \rightarrow \tau\), if \(T\) satisfies

\[
\forall x, y \in \text{ext}(\tau) \ (x[X] = y[X] \rightarrow x = y).
\]
That is, the set of values for attributes $X$ of an element (node) of type (label) $\tau$ uniquely identifies that element (node).

An absolute foreign key is an expression of the form $\tau_1[X] \subseteq_{FK} \tau_2[Y]$, where $\tau_1, \tau_2 \in E$, and $X, Y$ are nonempty lists of attributes in $R(\tau_1), R(\tau_2)$ of the same length. An XML tree $T$ satisfies the foreign key constraint $\tau_1[X] \subseteq_{FK} \tau_2[Y]$, written as $T \models \tau_1[X] \subseteq_{FK} \tau_2[Y]$, if $T$ satisfies the key constraint $\tau_2[Y] \rightarrow \tau_2$ and

$$\forall x \in ext(\tau_1) \exists y \in ext(\tau_2)(x[X] = y[Y]).$$

That is, the set of values for attributes $Y$ of an element of type $\tau_2$ uniquely identifies that element, and the list of values for attributes $X$ of an element of type $\tau_1$ has to match the list of values for attributes $Y$ of some element of type $\tau_2$.

Absolute keys and foreign keys define constraints that hold in the entire XML document. Since XML documents have hierarchical structure, we may also want to define constraints that hold within a specific part of an XML document. This gives rise to the definition of relative keys and foreign keys \cite{BDF01, AFL05} that we present below.

Given a DTD $D = (E, A, P, R, r)$, a relative key is an expression of the form $\tau(\tau_1[X] \rightarrow \tau_1)$, where $\tau, \tau_1 \in E$ and $X$ is a nonempty set of attributes in $R(\tau_1)$. Let $T$ be an XML tree conforming to $D$ and $x \prec y$ denote the descendant relationship between nodes $x$ and $y$ in $T$. Then $T$ satisfies the relative key constraint $\tau(\tau_1[X] \rightarrow \tau_1)$ if $T$ satisfies

$$\forall x \in ext(\tau) \forall y, z \in ext(\tau_1)((x \prec y) \land (x \prec z) \land (y[X] = z[X]) \rightarrow y = z).$$

Here $\tau$ is called the context type of the key constraint. This constraint specifies that for a fixed node $x$ of type $\tau$, the set of attributes $X$ is a key for nodes of type $\tau_1$ that are descendants of $x$. Obviously, absolute keys are special cases of relative keys, where the context type is the root. Also note that relative keys are very similar to the keys defined for weak entities in relational databases. Weak entities are entities that cannot be uniquely identified by their own attributes, but they become unique if we add to their attributes the key of another entity as a foreign key.
A relative foreign key is an expression of the form \( \tau(\tau_1[X] \subseteq_{FK} \tau_2[Y]) \), where \( \tau, \tau_1, \tau_2 \in E \), \( X, Y \) are nonempty lists of attributes in \( R(\tau_1), R(\tau_2) \) of the same length. An XML tree \( T \) satisfies \( \tau(\tau_1[X] \subseteq_{FK} \tau_2[Y]) \) if \( T \) satisfies the relative key constraint \( \tau(\tau_2[Y] \rightarrow \tau_2) \), and it also satisfies

\[
\forall x \in ext(\tau) \forall y \in ext(\tau_1)((x < y) \rightarrow \exists z \in ext(\tau_2) ((x < z) \land y[X] = z[Y])).
\]

This constraint specifies that for each \( x \) in \( ext(\tau) \), \( X \) is a foreign key of descendants of \( x \) of type \( \tau_1 \) that references a key \( Y \) of type \( \tau_2 \)-descendants of \( x \).

**Example 5.6** Consider the DTD shown in Example 5.4 and the XML tree in Figure 5.3 that conforms to the DTD. This tree satisfies the following key constraints. Note that the first constraint is a relative key.

\[
\begin{align*}
\text{branch[@bID]} & \rightarrow \text{branch} \\
\text{branch(client[@name]} & \rightarrow \text{client})
\end{align*}
\]

Given a DTD \( D \) and a set of key and foreign key constraints \( \Sigma \), there may be no XML tree conforming to \( D \) and satisfying \( \Sigma \), as observed by Fan and Libkin [FL01], due to the interaction of constraints imposed by \( D \) and \( \Sigma \). The *consistency* (or *satisfiability*) problem is a decision problem of determining whether there exists an XML document that conforms to \( D \) and satisfies \( \Sigma \). The problem is undecidable in general, where we have multi-attribute keys and foreign keys [FL02]. The complexity of this problem has been extensively studied [FL01, FL02, AFL02a, AFL05] for some restricted families of key and foreign key constraints.

### 5.2.2 Keys Defined by Path Expressions

We now present a more expressive key constraint that is based on the notion of *path expressions* and introduced by Buneman et al. [BDF+01b, BDF+01a]. A path expression
$P$ is a regular expression over a finite alphabet $\Sigma$ that is contained in $El \cup Att \cup \{S\} \cup \{\_\}$ defined by the following grammar:

$$P ::= \epsilon \mid a \mid P \cdot P \mid P \cup P \mid P^*,$$

where $\_\_$ represents a wild card matching any symbol in $El$, $\epsilon$ is the empty string, $a \in \Sigma$, and $\cdot$, $\cup$, and $^*$ denote concatenation, union, and Kleene star, respectively. A path expression $P$ is called simple if it does not contain $\_\_$ or $^*$. Given two nodes $x, y$ in XML tree $T$, we say that $y$ is reachable from $x$ by following $P$ if there is a string $w$ in the regular language defined by $P$ such that $w$ is a path from $x$ to $y$ in $T$. The set of all nodes reachable from $x$ by following $P$ is denoted by $x[J_P]$. 

A key specification is a pair $(Q, \{P_1, \ldots, P_k\})$, where $Q$ is a path expression and $\{P_1, \ldots, P_k\}$ is a set of simple path expressions. The notion of satisfaction of a key specification by an XML tree is defined using the notion of value equality [BDF+01a]. Given an XML tree $T$, two nodes in $T$ are value equal if they have the same label, and in addition, either they have the same string value or their children are pairwise value equal. A node $n$ satisfies a key specification $(Q, \{P_1, \ldots, P_k\})$ if for any $n_1, n_2$ in $n[Q]$, if for all $i \in [1, k]$, there exists $z_1 \in n_1[P_i]$ and $z_2 \in n_2[P_i]$ such that $z_1$ and $z_2$ are value equal, then $n_1 = n_2$.

Relative keys can also be expressed using the above notation. An XML tree $T$ satisfies a relative key specification $(Q, (Q', S))$ if for all nodes $n$ in $n[Q]$, $n$ satisfies the key $(Q', S)$. In other words, $(Q', S)$ is a key for every subdocument rooted at a node in $n[Q]$.

It is known that keys in this language are always finitely satisfiable, and the finite implication problem for keys is finitely axiomatizable and decidable in polynomial time in the size of the keys [BDF+01b, BDF+03]. Key specifications in this language do not assume the presence of any schema (e.g. DTD) for XML documents, and that is why reasoning about these keys can be done so efficiently.
5.3 Functional Dependencies for XML

Several languages have been proposed [AL04, VL03, VLL04b, LLL02, HL03, WT05] for expressing functional dependencies for XML. Most of them define a functional dependency as an expression of the form $p_1, \ldots, p_n \rightarrow q$, where $p_1, \ldots, p_n, q$ are path expressions. However, there are different definitions for satisfaction of a functional dependency by an XML tree. Here we briefly present some of these proposals but elaborate more on the one introduced by Arenas and Libkin [AL04] that will be used in the next chapters.

5.3.1 Functional Dependencies Defined by Tree Tuples

Arenas and Libkin [AL04] use a relational representation of XML documents to define the satisfaction of functional dependencies. This relational representation is based on the notion of tree tuples. Given an XML tree $T$ that conforms to a DTD $D$, a tree tuple [AL04] is intuitively a subtree of $T$ with the same root that contains at most one occurrence of every path. Then satisfaction is defined in the usual way: if two tree tuples in a tree agree on all the paths $p_1, \ldots, p_n$, then they must agree on $q$.

The precise definition requires a bit of care since a tree tuple may not be defined on some paths (as tree tuples have at most one occurrence of every path and may have zero occurrences). Let $\bot$ represent such missing values. Given a DTD $D = (E, A, P, R, r)$ and an XML tree $T = (V, \text{lab}, \text{ele}, \text{att}, \text{root})$ such that $T \models D$, a tree tuple $t$ in $T$ is formally defined as a function from $\text{paths}(D)$ to $\text{Vert} \cup \text{Str} \cup \{\bot\}$ such that if for an element path $q$ with last($q$) = $a$, we have $t(q) \neq \bot$, then

- $t(q) \in V$ and lab($t(q)$) = $a$;

- if path $q'$ is a prefix of path $q$, then $t(q') \neq \bot$ and $t(q')$ lies on the path from the root to $t(q)$ in $T$;

- if $@l$ is defined for $t(q)$ and its value is $s \in \text{Str}$, then $t(q.@l) = s$. 
A tree tuple \( t \) in \( T \) is maximal if there is no other tree tuple \( t' \) in \( T \) that is obtained by only replacing some null values in \( t \) with values from \( V \cup \text{Str} \). The set of maximal tree tuples in \( T \) is denoted by \( \text{tuples}_D(T) \).

**Example 5.7** Consider again the DTD \( D \) of Example 5.4 and the XML tree \( T \) in Figure 5.3. This tree contains four maximal tree tuples, one of which is shown in Figure 5.4. This tree tuple \( t \) can also be represented as a function.

\[
\begin{align*}
    t(\text{company}) &= v_0 \\
    t(\text{company}.\text{branch}) &= v_1 \\
    t(\text{company}.\text{branch}.@\text{bID}) &= \text{co201} \\
    t(\text{company}.\text{branch}.\text{type}) &= v_2 \\
    t(\text{company}.\text{branch}.\text{type}.S) &= \text{consulting} \\
    t(\text{company}.\text{branch}.\text{client}) &= v_3 \\
    t(\text{company}.\text{branch}.\text{client}.@\text{name}) &= \text{cl1} \\
    t(\text{company}.\text{branch}.\text{client}.\text{contact}) &= v_4 \\
    t(\text{company}.\text{branch}.\text{client}.\text{contact}.@\text{areaCode}) &= 416 \\
    t(\text{company}.\text{branch}.\text{client}.\text{contact}.@\text{phone}) &= 860 \text{ 1212} \\
    t(\text{company}.\text{branch}.\text{client}.\text{contact}.@\text{city}) &= \text{Toronto} \\
\end{align*}
\]

We now go back to the definition of functional dependencies. A functional dependency
(FD) over a DTD $D$ [AL04] is an expression of the form $\{q_1, \ldots, q_n\} \rightarrow q$, where $n \geq 1$ and $q, q_1, \ldots, q_n \in \text{paths}(D)$. An XML tree $T$ that conforms to $D$ satisfies an FD $\{q_1, \ldots, q_n\} \rightarrow q$, written as $T \models \{q_1, \ldots, q_n\} \rightarrow q$, if for any two tree tuples $t_1, t_2 \in \text{tuples}_D(T)$, whenever $t_1(q_i) = t_2(q_i) \neq \bot$ for all $i \in [1, n]$, then $t_1(q) = t_2(q)$.

**Example 5.8** The XML tree of Figure 5.3 satisfies the following constraints, all of which can be expressed as functional dependencies.

- The value of the @bID attribute is unique for each branch element:
  
  \[
  \text{company.branch.@bid} \rightarrow \text{company.branch}.
  \]

- Given a specific branch element, no two client elements can have the same value for @name:
  
  \[
  \text{company.branch, company.branch.client.@name} \rightarrow \text{company.branch.client}.
  \]

- The value of attribute @areaCode determines the value of attribute @city:
  
  \[
  \text{company.branch.client.contact.@areaCode} \rightarrow \text{company.branch.client.contact.@city}.
  \]

Given a DTD $D$ and a set $\Sigma \cup \{\varphi\}$ of FDs, we say that $\varphi$ is implied by $(D, \Sigma)$ if every XML tree $T$ that conforms to $D$ and satisfies $\Sigma$ also satisfies $\varphi$. The set of all FDs implied by $(D, \Sigma)$ is denoted by $(D, \Sigma)^+$. An FD is called trivial if it belongs to $(D, \emptyset)^+$. Remember that in relational databases an FD $X \rightarrow Y$ is trivial only if $Y \subseteq X$. However, more complicated trivial FDs can be inferred from a DTD and a set of FDs, due to the interaction of constraints imposed by DTDs with functional dependencies. For example, an FD $p \rightarrow q$, where $p$ is an element path and $q$ is a prefix of $p$, is trivial.

The implication problem for XML functional dependencies has been extensively studied [AL04], and here we present a brief summary of the results. In general, the implication
problem for XML functional dependencies over DTDs is solvable in co-NEXPTIME, and furthermore, the implication problem is not finitely axiomatizable. However, the implication problem can be solved in polynomial time for some restricted classes of DTDs, such as simple DTDs and disjunctive DTDs with bounded number of disjunctions. Below we define these classes.

A regular expression over an alphabet $A$ is called trivial if it is of the form $s_1, \ldots, s_n$, where each $s_i$ is either $a_i$, $a_i?$, $a_i^+$, or $a_i^*$ for some $a_i \in A$, such that for $i \neq j$, $a_i \neq a_j$. A regular expression $s$ is called simple [AL04] if there is a trivial regular expression $s'$ such that any word in the language of $s$ is a permutation of a word in the language of $s'$, and vice versa. Then simple DTDs are the ones in which all production rules use simple regular expressions over the alphabet of element names $El$. Another class of DTDs for which the implication problem is tractable is the class of DTDs in which all the production rules are made of a bounded number of disjunctions of simple regular expressions.

5.3.2 Other Definitions for Functional Dependencies

Vincent et al. [VL03, VLL04b] define XML functional dependencies based on paths and path instances, rather than tree tuples. The existence of a DTD is not assumed in this definition. Functional dependencies, called XFDs, are again expressions of the form $p_1, \ldots, p_n \rightarrow p$, where $p_1, \ldots, p_n, p$ are paths. The notion of strong satisfaction of an FD by an XML tree is defined using the agreement of path instances in the tree on the paths in the FD. This definition takes into account the missing values in the XML tree. It is shown that an XML document strongly satisfies an XFD if and only if every completion of the XML document by putting values for nulls satisfies the XFD. However, when there is no missing information in the XML document, the definition of Vincent et al. coincides with the definition of Arenas and Libkin [AL04]. The notion of local functional dependencies (LFDs) [LVL03] is introduced as a generalization of regular XFDs. They
are functional dependencies that hold only in a certain part of an XML document and not in the whole document. The implication problem of LFDs is briefly studied [LVL03], but the given set of axioms is not complete for reasoning about LFDs.

Another alternative definition of XML functional dependencies by Hartmann and Link [HL03] is based on homomorphism between vertices of XML data trees and XML schema graphs. In this approach the schema of XML documents is represented by schema trees, rather than DTDs. An XML schema tree and an XML data tree compatible with it are shown in Figures 5.5 and 5.6 respectively. Every edge in the schema tree has frequency one if it does not have a label, or frequency zero or more if it is labeled with symbol *. Given a schema tree $T$, a functional dependency (or XFD) is an expression of the form $v : X \rightarrow Y$, where $v$ is a vertex in $T$, and $X$ and $Y$ are subgraphs of $T$ rooted at $v$. Satisfaction of an FD by an XML data tree is defined using the concepts from graph theory such as subgraph, homomorphism, and projection. We do not present the details of definition of XFDs here, but we show an example of XML functional dependencies that can be expressed using this approach, but cannot be expressed with any of the definitions presented earlier. Consider the XML tree in Figure 5.6. Suppose this document satisfies the following constraint: whenever two courses agree on all their pairs, they agree on their rating too. Since the previous definitions of XML FDs [AL04, VL03] use path expressions, they are not able to express functional dependencies of this kind.

A more recent proposal for XML functional dependencies by Wang and Topor [WT05]
Figure 5.6: An XML data tree [HL03].

defines different types of agreement for two nodes on a path to make the FDs more expressive. We need some terminology before giving the definition of FDs as suggested by Wang and Topor. A \textit{downward path} is an expression of the form $l_1.l_2.\ldots.l_n$, where each $l_i$ is an element name, attribute name, symbol $\_\,$ representing wildcard, or symbol $\sim\,$ representing Kleene star. Note that $l_n$ can only be either an element or attribute name. A \textit{simple path} is a downward path that has no $\_\,$ or $\sim\,$. An upward path is of the form $\uparrow \ldots \uparrow$, and a \textit{composite path} is of the form $\psi.\rho$, where $\psi$ and $\rho$ are an upward path and a simple path respectively. The validity of every path with respect to an XML tree can also be defined in a straightforward way.

Recall that $n[p]$ denotes the set of nodes reached by starting at node $n$ and following the path $p$. Given an XML path $p$, two nodes $n_1, n_2$ in XML tree $T$ can agree on path $p$ in one of the following ways:

- We say $n_1, n_2$ \textit{node agree} or \textit{N-agree} on $p$ if
  - path $p$ is a simple path, and $n_1 = n_2$; or
  - path $p$ is an upward path, $n_1[p] = n_2[p]$; or
  - path $p$ is a composite path, and $n_1, n_2$ node agree on the upward path part of $p$.

- We say $n_1, n_2$ \textit{set agree} or \textit{S-agree} on $p$ if for every node $v_1$ in $n_1[p]$, there is a node
Figure 5.7: A school database [WT05].

\[ v_2 \in n_2[p] \text{ such that } v_1, v_2 \text{ are value equal, and vice versa.} \]

- We say \( n_1, n_2 \) intersect agree or I-agree on \( p \) if there exist nodes \( v_1 \in n_1[p] \) and \( v_2 \in n_2[p] \) such that \( v_1, v_2 \) are value equal.

Then Wang and Topor [WT05] define an XML functional dependency on XML tree \( T \) is an expression of the form \( Q : p_1(c_1), \ldots, p_n(c_n) \rightarrow p_{n+1}(c_{n+1}) \), where \( Q \) is a downward path, \( p_1, \ldots, p_n \) are simple or composite paths, \( p_{n+1} \) is a simple path of length 1 or 0, and \( c_i, i \in [1, n+1], \) is one of \( N, S, \) and \( I \). An XML tree \( T \) satisfies an FD if for any two nodes \( n_1, n_2 \in root[Q] \) we have the following: if \( n_1[p_i] \) is not empty, and \( n_1, n_2 \) \( c_i \)-agree on \( p_i \) for all \( i \in [1, n] \), then \( n_1[p_{n+1}] \) and \( n_2[p_{n+1}] \) are not empty, and \( n_1, n_2 \) also \( c_{n+1} \)-agree on \( p_{n+1} \). There are some constraints that can be expressed using the FDs defined by Wang and Topor, which are not expressible using other definitions of FDs by Arenas and Libkin [AL04] or Vincent et al. [VL03, VLL04b]. For example, consider the XML tree of Figure 5.7. To specify that any single telephone number of a student determines his/her set of addresses, we can write:

\( \sim .student : tel(I) \rightarrow address(S). \)
Although XML functional dependencies defined by Arenas and Libkin may not be as expressive as some FDs defined by others [VL03, HL03, WT05], we believe that they are the most natural generalization of relational FDs for XML. Moreover, they are expressive enough for the types of constraints that we will consider in the next chapters. Therefore, hereafter, by XML functional dependencies, we will only be referring to the tree tuple FDs.

### 5.4 Designing XML Data

Database design and normalization is a fundamental and well-studied subject with a long history in relational databases (see an old survey by Beeri et al. [BBG78]). Given a schema and a set of data dependencies, the goal is to refine the schema into a better schema so that update, insertion, or deletion anomalies are less likely. Anomalies can happen for XML data as well, if the schemas (e.g., DTDs) are not well-designed. The design problem for XML data has recently been a subject of interest for database researchers, and a number of normal forms have been proposed to avoid redundancies caused by functional [AL04, VLL04b, WT05, EM01, WLLD01] or multivalued [VLL03, EM01] dependencies. Here, we review a normal form proposed by Arenas and Libkin [AL04], which is a natural generalization of BCNF for XML data.

#### 5.4.1 XNF: A Normal Form for XML Documents

Arenas and Libkin [AL04] define a normal form for XML that does not allow any redundancy in data values occurring in the leaves of an XML tree, provided that the set of constraints consists of only functional dependencies.

Given a DTD $D$ and a set of FDs $\Sigma$ over $\text{paths}(D)$, $(D, \Sigma)$ is in XML normal form (XNF) if and only if for every nontrivial FD $\varphi \in (D, \Sigma)^+$ of the form $X \rightarrow p.@l$ or $X \rightarrow p.S$, the FD $X \rightarrow p$ is also in $(D, \Sigma)^+$. Intuitively, this normal forms ensures
that the value of \( p_1.@l \) will not be repeated in two different locations of the XML tree, corresponding to two tree tuples that share the same values for paths in \( X \). Therefore, there will not be any redundancy in the leaves of the tree that store string values for attributes or elements.

**Example 5.9** Referring back to Example 5.8, the following two FDs satisfy the XNF condition:

\[
\text{company.branch.@bid} \rightarrow \text{company.branch},
\]

\[
\text{company.branch, company.branch.client.@name} \rightarrow \text{company.branch.client},
\]

while the following FD does not:

\[
\text{company.branch.client.contact.@areaCode} \rightarrow \text{company.branch.client.contact.@city}.
\]

By looking at the XML tree in Figure 5.3 we realize that this FD may cause redundancies in the document by storing duplicates of the pairs \( \text{areaCode,city} \) in different places of the document.

To show that XNF is really a good replacement for BCNF, Arenas and Libkin [AL04] present a straightforward translation of relations into XML documents and show that the relation schema is in BCNF if and only if the corresponding XML representation satisfies XNF. They also give a more powerful justification for XNF by applying the information-theoretic measure introduced in Chapter 3 to this normal form. We will see a summary of the results in Section 5.4.3.

### 5.4.2 XNF Normalization and Dependency Preservation

Given an XNF design that is not in XNF, there is a normalization algorithm [AL04] that transforms that design into an XNF design. This algorithm assumes that the DTD is not recursive (the regular expressions are non-recursive), all the FDs are of the form \( \{q, p_1.@l_1, \ldots, p_n.@l_n\} \rightarrow p \) (they contain at most one element path on the left-hand
Figure 5.8: XNF decomposition operations: (a) Moving attributes, (b) Creating new element types [AL04].

Given a DTD $D = (E, A, P, R, r)$, and a set of FDs $\Sigma$, a nontrivial FD $S \rightarrow p.@l$ is called anomalous over $(D, \Sigma)$ if it violates XNF. That is, $S \rightarrow p.@l \in (D, \Sigma)^+$, but $S \rightarrow p \notin (D, \Sigma)^+$. A path on the right-hand side of an anomalous FD is called an anomalous path. We say that $\{q, p_1.@l_1, \ldots, p_n.@l_n\} \rightarrow p_0.@l_0$ is $(D, \Sigma)$-minimal if there is no anomalous FD $S' \rightarrow p_i.@l_i \in (D, \Sigma)^+$ such that $i \in [0, n]$ and $S'$ is a subset of $\{q, p_1, \ldots, p_n, p_0.@l_0, \ldots, p_n.@l_n\}$ such that $|S'| \leq n$ and $S'$ contains at most one element path.

The normalization algorithm, shown in Figure 5.9 iteratively performs the following two transformations to finally output an XML specification in XNF:

1. **Moving attributes**: Consider an anomalous FD $q \rightarrow p.@l \in (D, \Sigma)^+$, where $q$ is an element path. To eliminate it, the attribute $@l$ is moved from its original location

---

1This is a reasonable restriction since it is not easy to come up with a meaningful FD with more than one element path on the left-hand side.
Input: XML specification \((D, \Sigma)\).
Output: \((D, \Sigma)\) in XNF.

1. If \((D, \Sigma)\) is in XNF then return \((D, \Sigma)\), otherwise go to step (2).

2. If there is an anomalous FD \(X \rightarrow p.\@l\) and \(q \in EPaths(D)\) such that \(q \in X\) and \(q \rightarrow X \in (D, \Sigma)^+\), then:
   
   2.1. Choose a fresh attribute @\(m\)
   2.2. \(D := D[p.\@l := q.@m]\)
   2.3. \(\Sigma := \Sigma[p.\@l := q.@m]\)
   2.4. Go to step (1)

3. Choose a \((D, \Sigma)\)-minimal anomalous FD \(X \rightarrow p.\@l\), where \(X = \{q, p_1.\@l_1, \ldots, p_n.\@l_n\}\)
   
   3.1. Create fresh element types \(\tau, \tau_1, \ldots, \tau_n\)
   3.2. \(D := D[p.\@l := q.\tau[\tau_1.\@l_1, \ldots, \tau_n.\@l_n]]\)
   3.3. \(\Sigma := \Sigma[p.\@l := q.\tau[\tau_1.\@l_1, \ldots, \tau_n.\@l_n]]\)
   3.4. Go to step (1)

Figure 5.9: XNF decomposition algorithm [AL04].

to the set of attributes of the last element of \(q\), as shown in Figure 5.8(a). Formally, the new DTD \(D'\), written as \(D[p.\@l := q.@m]\), is defined to be \((E, A', P', R', r)\), where \(A' = A \cup \{@m\}\), \(R'(last(q)) = R(last(q)) \cup \{@m\}\), \(R'(last(p)) = R(last(p)) - \{@l\}\), and \(R'(\tau) = R(\tau)\) for each element \(\tau \in E - \{last(q), last(p)\}\). The new set of functional dependencies, written as \(\Sigma[p.\@l := q.@m]\), consists of all \(S_1 \rightarrow S_2\) with \(S_1 \cup S_2 \subseteq paths(D[p.\@l := q.@m])\).

2. Creating new element types: Consider an anomalous FD \(\{q, p_1.\@l_1, \ldots, p_n.\@l_n\} \rightarrow p.\@l \in (D, \Sigma)^+\), where \(q\) is an element path. To eliminate it, a new element \(\tau\) is created as a child of \(last(q)\), new elements \(\tau_1, \ldots, \tau_n\) are created as children of \(\tau\), the attributes \(\@l_1, \ldots, \@l_n\) are copied for \(\tau_1, \ldots, \tau_n\), and \(\@l\) is moved from its original location to become an attribute of \(\tau\), as shown in Figure 5.8(b). The new DTD \(D'\), written as \(D[p.\@l := q.\tau[\tau_1.\@l_1, \ldots, \tau_n.\@l_n]]\), is defined to be \((E', A', P', R', r)\), where:
The new set of FDs, written as $\Sigma[p.\cdot \@l := q.\cdot \tau[\tau_1.\cdot \@l_1, \ldots, \tau_n.\cdot \@l_n]]$, consists of all $S_1 \rightarrow S_2$ with $S_1 \cup S_2 \subseteq \text{paths}(D')$. Also, each FD over $q, p_1, \ldots, p_n, p_1.\cdot \@l_1, \ldots, p_n.\cdot \@l_n$ and $p.\cdot \@l$ is transferred to $\tau$ and $\tau_1, \ldots, \tau_n$. The FD set also contains $\{q, q.\cdot \tau_1.\cdot \@l_1, \ldots, q.\cdot \tau_n.\cdot \@l_n\} \rightarrow q.\cdot \tau$ and $\{q.\cdot \tau, q.\cdot \tau_i.\cdot \@l_i\} \rightarrow q.\cdot \tau_i$ for $i \in [1, n]$. To ensure that there will not be any anomalous FD in the structure formed by new element types $\tau, \tau_1, \ldots, \tau_n$, the transformation creating new element types is only applied to $(D, \Sigma)$-minimal anomalous FDs.

To show that the XNF decomposition algorithm does not lose any information, Arenas and Libkin [AL04] define the notion of lossless decomposition for DTDs and prove that if $(D', \Sigma')$ is obtained from $(D, \Sigma)$ by using one of the transformations from the decomposition algorithm, then $(D', \Sigma')$ is a lossless decomposition of $(D, \Sigma)$. However, the concept of dependency preservation for XML is not addressed by Arenas and Libkin. In fact, there is still no clear understanding of dependency preservation for XML decompositions. We informally show in the following example that XNF normalization may lose some functional dependencies.

**Example 5.10** We saw in Example 5.9 that the anomalous FD

$$\text{company.branch.client.contact.}@\text{areaCode} \rightarrow \text{company.branch.client.contact.}@\text{city}$$

causes redundancies in the document of Figure 5.3. To remove this FD, the decomposition algorithm restructures the document by creating new element types, as shown in Figure 5.10. Now consider again the original document in Figure 5.3 that satisfies the following constraint: if two clients are in the same city and require a certain type of
service, they are under the same company branch. This constraint can be written as the following FD over the original DTD of Example 5.4:

\[
\{\text{company.branch.client.contact.@city, company.branch.type}\} \rightarrow \text{company.branch}.
\]

This FD does not hold in the restructured document in Figure 5.10, because by splitting the cities information, we have broken the nesting of element \text{branch} and attribute \text{city}. In this example, we observe that the XNF decomposition algorithm is not dependency-preserving.

5.4.3 Justifying Perfect XML Normal Forms

The notion of being well-designed, introduced in Chapter 3, can also be applied to XML databases. We first need to define the set of positions in an XML tree. It is intuitively the set of places where values (i.e., attribute values) occur. Formally, for a tree \( T = (V, \text{lab}, \text{ele}, \text{att}, \text{root}) \) that conforms to DTD \( D \), the set of positions \( \text{Pos}(T) \) is defined as the set \( \{(x, @l) \mid x \in V, \text{att}(x, @l) \text{ is defined}\} \). It is again assumed that the domain of all variables is a countably-infinite set. For a position \( p \in \text{Pos}(T) \), the relative information content \( \text{Ric}_T(p \mid \Sigma) \) is defined exactly as it is defined for the relational case [AL05]. Let \( \text{inst}(D, \Sigma) \) denote the set of all XML trees that conform to a DTD \( D \) and satisfy FDs \( \Sigma \). Then, a well-designed XML specification is defined as follows [AL05]:

Figure 5.10: A redundancy-free XML document.
Definition 5.11  An XML specification $(D, \Sigma)$ is well-designed if for every $T \in \text{inst}(D, \Sigma)$ and every $p \in \text{Pos}(T)$, $\text{RIC}_T(p \mid \Sigma) = 1$.

The following result shows that normal form XNF precisely characterizes well-designed XML documents that contain functional dependencies.

Theorem 5.12 (Arenas and Libkin [AL05]) Let $D$ be a DTD and $\Sigma$ be a set of FDs over $D$. Then $(D, \Sigma)$ is well-designed if and only if it is in XNF.

The XNF decomposition algorithm, presented in Section 5.4.2, is also justified using the information-theoretic measure by showing that the algorithm never decreases the amount of information content of a data value in any of its steps [AL05].
Chapter 6

A Redundancy-Free XML Design for Relational Data

Having a database design that avoids redundant information and update anomalies is the main goal of normalization techniques. Ideally, data as well as constraints should be preserved. However, this is not always achievable: while BCNF eliminates all redundancies, it may not preserve constraints, and 3NF, which achieves dependency preservation, may not always eliminate all redundancies. In this chapter, we study the possibility of achieving both redundancy elimination and dependency preservation by a hierarchical representation of relational data in XML. We provide a characterization of cases when we can guarantee both by designing an XML document that is in the normal form XNF, and present an algorithm that produces such a design. The goal of this chapter is to provide guidelines on how to design an XML view of a relational database that preserves all the relational functional dependencies and is also free of redundancy.

6.1 Introduction

In relational database design, relations are sometimes normalized to avoid redundancies and update anomalies. Losslessness, dependency preservation, and redundancy elimi-
nation are three desired properties for every normalization. However, it is not always possible to achieve all three: while BCNF normalization eliminates all redundancies, it may not preserve dependencies, and 3NF normalization, which achieves dependency preservation, may sometimes produce relations that store data with high degrees of redundancy, as we have seen in Chapter 4.

In this chapter, we present recent results [Kol05, Kol07] showing that for some relations, it is possible to produce a dependency-preserving XML representation, which is in XNF and hence avoids redundancies and update anomalies. One can therefore take advantage of the good properties of BCNF and 3NF that may not be achievable together in relational representations.

**Example 6.1** Consider the relation schema \( R(A, B, C) \), with FDs \( \mathcal{F} = \{AB \rightarrow C, C \rightarrow B\} \). This is a classical example of a schema in 3NF that does not have any FD-preserving BCNF decomposition. We can design a DTD \( D = (E, A, P, R, r) \) and a set of XML functional dependencies \( \Sigma \) over \( D \), such that \((D, \Sigma)\) is in XNF, where

- \( E = \{r, A, B, C\} \).
- \( A = \{@a, @b, @c\} \).
- \( P(r) = B^*, \ P(B) = A^*, \ P(A) = C^*, \ P(C) = \epsilon. \)
- \( R(r) = \emptyset, \ R(A) = \{@a\}, \ R(B) = \{@b\}, \ R(C) = \{@c\}. \)
- \( \Sigma = \{r.B.@b \rightarrow r.B, \{r.B, r.B.A.@a\} \rightarrow r.B.A, \{r.B.A, r.B.A.C.@c\} \rightarrow r.A.B.C, \{r.B.A.@a, r.B.@b\} \rightarrow r.B.A.C.@c, r.B.A.C.@c \rightarrow r.B.@b\}. \)

Note that \( \epsilon \) is the empty regular expression. This conversion is visualized in Figure 6.1.

The first three FDs are the result of the nested structure of the document. The second
FD specifies that given a B element, a value of attribute @a uniquely determines one of the children, which is an A element. The last two FDs are the translations of relational FDs in \( \mathcal{F} \). From the set of FDs \( \Sigma \), we can easily infer the following two FDs: 
\[
\{ r.B.A.@a, r.B.@b \} \rightarrow r.B.A.C \quad \text{and} \quad r.B.A.C.@c \rightarrow r.B. 
\]
Thus, \( (D, \Sigma) \) is in normal form XNF and hence does not allow redundancy.

In the above example, the correct hierarchical ordering of elements in the DTD makes it possible to have an XML representation in XNF from a non-BCNF relation. Since each relational FD can be enforced by a corresponding XML FD, the representation is also dependency-preserving. The XML design \( (D, \Sigma) \) is therefore a dependency-preserving redundancy-free representation of relation schema \( (\mathcal{R}, \mathcal{F}) \).

We formally define the concept of hierarchical XML representation for an arbitrary relation schema in Section 6.2 and present the necessary and sufficient conditions for a relation schema to have a hierarchical XML representation in XNF. In Section 6.3, we give an algorithm that finds an XNF representation for a relation schema, if possible. Finally, in Section 6.4, we introduce a more general XML representation for relations, namely, semi-hierarchical translation.
6.2 Dependency-Preserving Hierarchical Representation of Relations

Now we formally define how to translate a relational design, which consists of a relation schema and a set of functional dependencies, into a hierarchical XML design consisting of a DTD and a set of XML functional dependencies. We then show that under what conditions this XML representation would be in normal form XNF, corresponding to redundancy-free XML documents.

Definition 6.2 Let $\mathcal{R}(A_1, \ldots, A_m)$ be a relation schema and $\mathcal{F}$ be a set of FDs defined over it. We define hierarchical translation of $(\mathcal{R}, \mathcal{F})$ to be the XML specification $(D, \Sigma)$, where $D = (E, A, P, R, r)$, $\Sigma$ is a set of XML functional dependencies over $D$, and

- $E = \{\tau_1, \ldots, \tau_m\} \cup \{r\}$ (each element type $\tau_i$ corresponds to a relational attribute $A_i \in \text{sort}(\mathcal{R})$);
- $A = \{@l_1, \ldots, @l_m\}$;
- $R(r) = \emptyset$ and for $i \in [1, m]$, $R(\tau_i) = \{@l_i\}$ (each element has an attribute to store a value);
- element types in $E$ form an ordering $\tau_{\pi_1}, \ldots, \tau_{\pi_m}$ such that $P(r) = \tau_{\pi_1}^*$, $P(\tau_{\pi_m}) = \epsilon$, and for every $i \in [1, m]$, $P(\tau_{\pi_i}) = \tau_{\pi_{i+1}}^*$;
- the FD $r.\tau_{\pi_1}.@l_{\pi_1} \rightarrow \tau_{\pi_1}$ is in $\Sigma$; moreover, for each $i \in [2, m]$, there is an FD $\{p, p.\tau_{\pi_i}.@l_{\pi_i}\} \rightarrow p.\tau_{\pi_i}$ in $\Sigma$, where $p$ is the path from the root to the parent of $\tau_{\pi_i}$;
- for every FD $X_1 \rightarrow X_2 \in \mathcal{F}$, there is a corresponding FD $S_1 \rightarrow S_2 \in \Sigma$, such that for every attribute $A_i$ in $X_1$ ($X_2$), there is a path $p.\tau_i.@l_i$ in $S_1$ ($S_2$), where $\tau_i$ corresponds to $A_i$ and $p$ is the path from the root to the parent of $\tau_i$. 
It is easy to see that this translation is dependency-preserving in the following sense: suppose we translate relation schema \((R, \mathcal{F})\) into a hierarchical XML specification \((D, \Sigma)\). Then every instance \(I\) of \(R\) gets translated into an XML tree \(T\) that conforms to \(D\). Now \(I\) satisfies \(\mathcal{F}\) if and only if \(T\) satisfies \(\Sigma\), and hence none of the FDs are lost.

Note also that the hierarchical translation of a relation schema is somewhat similar to a tree representation of a nesting of the schema. The inverse transformation, therefore, would simply be the unfolding of the nested structure. In other words, given an XML tree \(T\) conforming to DTD \(D\) and satisfying FDs \(\Sigma\), where \((D, \Sigma)\) is a hierarchical translation of \((R, \mathcal{F})\), \(T\) can be easily unfolded into an instance \(I\) of \(R\) that satisfies \(\mathcal{F}\).

Let \(R\) be a relation schema with a set of FDs \(\mathcal{F}\) and \((D, \Sigma)\) be a hierarchical translation of \((R, \mathcal{F})\), where \(D = (E, A, P, R, r)\). The following theorem defines when this XML representation is free of redundancies, or equivalently, when \((D, \Sigma)\) is in XNF:

**Theorem 6.3** The XML specification \((D, \Sigma)\) is in XNF if and only if for every FD \(X \rightarrow p.@l \in (D, \Sigma)^+\) and every prefix \(q\) of path \(p\), it is the case that \(X \rightarrow q.@m \in (D, \Sigma)^+\), where \(R(\text{last}(q)) = \{@m\}\).

*Proof:* \((\Rightarrow)\) Suppose \((D, \Sigma)\) is in XNF and the FD \(X \rightarrow p.@l \in (D, \Sigma)^+\). Then the FD \(X \rightarrow p\) is also in \((D, \Sigma)^+\). Let \(T\) be an arbitrary XML tree conforming to \(D\) and satisfying \(\Sigma\). For every two tree tuples \(t_1, t_2\) in \(T\), if \(t_1\) and \(t_2\) agree on all paths in \(X\) \((t_1(q') = t_2(q') \neq \perp\) for all \(q' \in X\)), then \(t_1(p) = t_2(p) = v\). Trivially, \(t_1\) and \(t_2\) agree on every node \(v'\) that is an ancestor of \(v\) and the attributes defined for \(v'\). Thus for every path \(q\) that is a prefix of \(p\) and every attribute \(@m\) defined for \(\text{last}(q)\), \(T\) satisfies the FDs \(X \rightarrow q\) and \(X \rightarrow q.@m\). Therefore, \(X \rightarrow q.@m\) is in \((D, \Sigma)^+\).

\((\Leftarrow)\) The proof of the other direction follows from the FDs resulting from the hierarchical representation of the relational attributes. Suppose an FD \(X \rightarrow p.@l \in (D, \Sigma)^+\). Then for every prefix \(q\) of \(p\) and the attribute \(@m\) defined for \(\text{last}(q)\) the FD \(X \rightarrow q.@m\) is also in \((D, \Sigma)^+\). Let \(T\) be an arbitrary XML tree conforming to \(D\) and satisfying \(\Sigma\). If
two tree tuples \( t_1, t_2 \) in \( T \) agree on all the attributes of elements from the root to \( \text{last}(p) \), they will agree on the nodes corresponding to element types from the root to \( \text{last}(p) \) as well. This is because of the FDs of the form \( \{p, p.\tau_i.\@l_i\} \rightarrow p.\tau_i \) that are added to \( \Sigma \) during the construction of \((D, \Sigma)\). Therefore, for every path \( q \) that is a prefix of \( p \), the FD \( X \rightarrow q \) is in \((D, \Sigma)^+\). In particular, \( X \rightarrow p \in (D, \Sigma)^+ \), and hence \((D, \Sigma)\) is in XNF. □

Since every attribute path \( p.\@l \in \text{paths}(D) \) represents exactly one relational attribute, the condition of Theorem 6.3 for \((D, \Sigma)\) to be in XNF translates to the following condition for \((\mathcal{R}, \mathcal{F})\).

**Corollary 6.4** A relation schema \((\mathcal{R}, \mathcal{F})\) has an XNF (redundancy-free) hierarchical translation into XML if and only if we can put the attributes of \( \mathcal{R} \) in order \( A_{\pi_1}, \ldots, A_{\pi_m} \) such that for every nontrivial FD \( X \rightarrow A_{\pi_i} \in \mathcal{F}^+ \) and every \( j < i \), the FD \( X \rightarrow A_{\pi_j} \) is also in \( \mathcal{F}^+ \).

**Example 6.5** Consider the following functional dependencies over the relation schema \( \mathcal{R}(A, B, C, D, F) \):

\[
ABCD \rightarrow F \\
FD \rightarrow A \\
FC \rightarrow B
\]

Since none of the FDs \( FC \rightarrow A \) or \( FD \rightarrow B \) holds, we cannot put the attributes in the desired order, and the schema does not have an XNF hierarchical translation. □

Some relation schemas do not satisfy the condition of Corollary 6.4. However, there might be a dependency-preserving decomposition of them such that each of the decomposed schemas can be hierarchically translated into an XML specification in XNF. Assume that we have found XML representations in XNF, denoted by \((D_1, \Sigma_1), \ldots, (D_n, \Sigma_n)\), each of which corresponding to one of the decomposed relations. We can combine all the DTDs into a single DTD by concatenating all the regular expressions assigned to their
roots, and assign the resulting expression to a new root. Then we have to take the union of element types, attributes, and FDs. Note that we assume the sets of element types of \(D_1, \ldots, D_n\) are disjoint. This can be seen in the following example and is formally described in Section 6.3. It is easy to observe that the combined DTD will not violate XNF since every \((D_i, \Sigma_i), i \in [1, n]\), is in XNF, and the sets \(\text{paths}(D_1), \ldots, \text{paths}(D_n)\) are disjoint.

**Example 6.6** Consider the following set \(\mathcal{F}\) of functional dependencies over the relation schema \(\mathcal{R}(A, B, C, D, F, G)\):

\[
\begin{align*}
ABCD & \rightarrow FG \\
DF & \rightarrow A \\
G & \rightarrow B \\
DG & \rightarrow A
\end{align*}
\]

Consider the following dependency-preserving decomposition for this schema consisting of two relations: \(\mathcal{R}_1(A, B, C, D, F)\) with FDs \(\mathcal{F}_1 = \{ABCD \rightarrow F, DF \rightarrow A\}\), and \(\mathcal{R}_2(A, B, C, D, G)\) with FDs \(\mathcal{F}_2 = \{ABCD \rightarrow G, G \rightarrow B, DG \rightarrow A\}\). We can put the attributes of \(\mathcal{R}_1\) and \(\mathcal{R}_2\) in the following orders: \(A, D, F, B, C\) and \(B, G, A, D, C\), which satisfy the condition of Corollary 6.4. Then a possible XML representation in XNF would include DTD \(D = (E, A, P, R, r)\) as follows. The set of FDs \(\Sigma\), omitted here, consists of the FDs resulting from the hierarchical translation and the FDs corresponding to the relational FDs. It can be easily verified that \((D, \Sigma)\) is in XNF.

- \(E = \{r, A, B, C, D, F, A', B', C', D', G'\}\).

- \(A = \{@a, @b, @c, @d, @f, @g\}\).

- \(P(r) = A^*B'^*, \quad P(A) = D^*, \quad P(D) = F^*, \quad P(F) = B^*, \quad P(B) = C^*, \quad P(C) = \epsilon, \quad P(B') = G'^*, \quad P(G') = A'^*, \quad P(A') = D'^*, \quad P(D') = C'^*, \quad P(C') = \epsilon.\)
**Input:** Relation schema \( R \) and set of FDs \( F \).

**Output:** Either \((D, \Sigma)\) in XNF or “No XNF Representation”.

Initialize \((D, \Sigma)\) with only a root \( r \);
Compute \( F_{\text{min}} \) as a minimal cover of \( F \);
Let \((R_1, F_1), \ldots, (R_n, F_n)\) be a lossless dependency-preserving 3NF+ decomposition of \((R, F)\) based on \( F_{\text{min}} \);

\[
\text{for } i := 1 \text{ to } n \text{ do}
\]

if there is no FD in \( F_i \) then

\[
O_i := \text{an arbitrary ordering of attributes in } R_i;
\]
else

\[
\text{Compute } X_i^{j^+} \text{ for all FD } X_i^j \rightarrow A_i^j \text{ in } F_i;
\]
\[
X := \text{attributes in } R_i;
\]
\[
O_i := \text{empty ordering};
\]

while \( X \neq \emptyset \) do

if no FD in \( F_i \) has an attribute in \( X \) on the right-hand side then

\[
Y := X;
\]
else

\[
Y := ( \bigcap_{X^j \in X} X_i^{j^+} ) \cap X;
\]

if \( Y = \emptyset \) then

\[
\text{return } \text{“No XNF Representation”};
\]
else

\[
\text{Append attributes in } Y \text{ to the ordering } O_i;
\]
\[
X := X - Y;
\]
\[
(D, \Sigma) := \text{attach } ((D, \Sigma), O_i, F_i);
\]

return \((D, \Sigma)\);

---

Figure 6.2: Dependency-preserving translation of relational data into redundancy-free XML documents.

- \( R(r) = \emptyset \), \( R(A) = R(A') = \{ @a \} \), \( R(B) = R(B') = \{ @b \} \),
- \( R(C) = R(C') = \{ @c \} \), \( R(D) = R(D') = \{ @d \} \), \( R(F) = \{ @f \} \),
- \( R(G) = \{ @g \} \).

Note that the original schema does not have a hierarchical translation in XNF since neither of \( DF \rightarrow B \) or \( G \rightarrow A \) holds, so the decomposition is necessary. We will see in the next section how we come up with the above hierarchical orderings. \( \square \)
6.3 Algorithm

Let $F_{\text{min}} = \{X_1 \rightarrow A_1, \ldots, X_k \rightarrow A_k\}$ denote the minimal cover of the FDs over the relation schema $R$. In order to find the appropriate order of attributes, we shall first compute the intersection of the closures of all $X_i$’s ($i \in [1, k]$). If the intersection is empty, there is no hierarchical translation in XNF for this relation schema. Otherwise, we output the attributes in the intersection in an arbitrary order as the first elements of the ordering. We remove from $F_{\text{min}}$ the FDs whose right-hand sides are already in the output. Then, we repeat computing the intersection of closures of the left-hand sides of the remaining FDs until there is no FD left or all the attributes are in the output. This procedure is described in the algorithm in Figure 6.2. Note that if $F_{\text{min}} = \emptyset$, we can output the attributes of $R$ in any arbitrary order.

Given a relation schema $(R, F)$, the algorithm in Figure 6.2 decides whether there is a redundancy-free hierarchical XML representation for it and produces an XML specification $(D, \Sigma)$ if there is one. First, the minimal cover of the FD set is computed. Then the 3NF synthesis algorithm that produces indecomposable schemas (see Figure 4.1) is applied to the minimal cover to increase the chance of having an XNF representation to the possible extent. The algorithm then finds the right ordering of attributes for each decomposed relation as described previously. Once an ordering is found, it should be attached to the DTD that is being constructed incrementally. This is done by the operator attach, which given an XML specification $(D, \Sigma)$, an ordering of relational attributes $O_i$ and a set of FDs $F_i$ over it, updates $(D, \Sigma)$ by performing the following steps:

- For each $j \in [1, |O_i|]$, create a fresh element type $\tau_j$ corresponding to $j$th attribute in $O_i$ and a fresh attribute name $@l_j$. Then assign $R(\tau_j) := \{@l_j\}$.
- Update $P(r) := P(r).\tau_1^*$, and for each $j \in [1, |O_i|)$, assign $P(\tau_j) := \tau_{j+1}^*$. Assign $P(\tau_{|O_i|}) := \epsilon$.
- Add to $\Sigma$ the FDs $r.\tau_1.\@l_1 \rightarrow r.\tau_1$ and $\{p, p.\tau_j.\@l_j\} \rightarrow p.\tau_j$ for each $j \in (1, |O_i|]$,
\( \mathcal{R} = (A, B, C, D, F, G) \)
\( \mathcal{F} = \{ABCD \rightarrow FG, DF \rightarrow A, G \rightarrow B, DG \rightarrow A\} \)

\( \mathcal{R}_1 = (A, B, C, D, F) \)
\( \mathcal{F}_1 = \{ABCD \rightarrow F, DF \rightarrow A\} \)

1. \( (ABCD)^+ \cap (DF)^+ = \{A, D, F\} \)
   ordering: \( A, D, F, \ldots \)
2. no FDs left.
   ordering: \( A, D, F, B, C \)

\( \mathcal{R}_2 = (A, B, C, D, G) \)
\( \mathcal{F}_2 = \{ABCD \rightarrow G, G \rightarrow B, DG \rightarrow A\} \)

1. \( (ABCD)^+ \cap G^+ \cap (DG)^+ = \{B, G\} \)
   ordering: \( B, G, \ldots \)
2. \( (DG)^+ = \{A, B, D, G\} \)
   ordering: \( B, G, A, D, \ldots \)
3. no FDs left.
   ordering: \( B, G, A, D, C \)

Figure 6.3: Finding the orderings of attributes using the algorithm in Section 6.3.

where \( p \) is the path from the root to the parent of \( \tau_j \).

- Translate each FD in \( \mathcal{F}_i \) into an XML FD by finding the corresponding DTD elements and add it to \( \Sigma \).

The most expensive step of the algorithm in Figure 6.2 is finding the minimal cover of a set of functional dependencies \( \mathcal{F} \), which could be done in polynomial time. More precisely, the minimal cover can be computed in \( O(|\mathcal{F}|^2) \), where \( |\mathcal{F}| \) is the length of the representation of \( \mathcal{F} \) in the number of attributes [BB79]. Therefore, the algorithm in Figure 6.2 runs in quadratic time in the length of the set of FDs to produce a redundancy-free XML representation for a given relational schema.

**Example 6.7** Figure 6.3 shows how the algorithm works on relation schema \((\mathcal{R}, \mathcal{F})\) of Example 6.6. The two FDs written in bold violate the condition of Corollary 6.4, so we cannot find the ordering directly and need a decomposition.
6.4 A More General Representation

So far we have considered a special way of converting relational data into a tree-like XML document, namely hierarchical translation. When there is no redundancy-free hierarchical XML representation for a relation schema \((R, \mathcal{F})\), one might think of other ways of translating \((R, \mathcal{F})\) into an XML specification \((D, \Sigma)\), such that \((D, \Sigma)\) is in XNF and hence does not allow redundancy. Here, we claim that when \((R, \mathcal{F})\) cannot be converted into a redundancy-free hierarchical XML specification, even a more general approach, named semi-hierarchical translation, does not help.

In this approach, instead of having one element type per relational attribute, we allow element types to represent more than one relational attribute. The formal definition of semi-hierarchical translation would be similar to that of hierarchical translation. The only difference would be that the attribute set assigned to each element type would no longer be a singleton. The FDs added as the result of the hierarchical structure would also reflect this change; they may have more than one attribute path on the left-hand side.

Example 6.8 A semi-hierarchical representation for relation schema of Example 6.1 consists of DTD \(D = (E, A, P, R, r)\) and a set of FDs \(\Sigma\) as follows:

- \(E = \{r, AB, C\}\).
- \(A = \{@a, @b, @c\}\).

- \(P(r) = AB^*, P(AB) = C^*, P(C) = \epsilon\).
- \(R(r) = \emptyset, R(AB) = \{@a, @b\}, R(C) = \{@c\}\).
- \(\Sigma = \{\{r.AB.@a, r.AB.@b\} \rightarrow r.AB,\)
  \(\{r.AB, r.AB.C.@c\} \rightarrow r.AB.C,\)
  \(\{r.AB.@a, r.AB.@b\} \rightarrow r.AB.C.@c,\)
  \(r.AB.C.@c \rightarrow r.AB.@b\}\).
In this translation, the element type AB represents two relational attributes A and B and therefore needs two attributes @a and @b.

The following theorem shows that checking the conditions of Corollary 6.4 is enough to know whether or not a relation schema has a non-redundant semi-hierarchical XML representation.

**Theorem 6.9** A relation schema \((R, \mathcal{F})\) has a redundancy-free hierarchical translation if and only if it has a redundancy-free semi-hierarchical translation.

**Proof:** One direction of the proof is trivial. It is enough to show that if \((R, \mathcal{F})\) has a semi-hierarchical XML translation in XNF, then we can construct a hierarchical XML representation, which is also in XNF. Let \((D, \Sigma)\) be a semi-hierarchical translation of \((R, \mathcal{F})\) in XNF. The non-root element types in \(D\) form an ordering \(\tau_1, \ldots, \tau_l\) from the root to the leaves. For instance, the ordering corresponding to DTD in Example 6.8 is \(AB, C\). Each element type corresponds to one or more than one relational attribute. We construct the corresponding ordering of relational attributes as follows: for each element type \(\tau_i\) representing \(k, k \geq 1\), relational attributes \(A_{i_1}, \ldots, A_{i_k}\), we output an arbitrary ordering \(A_{\pi_{i_1}}, \ldots, A_{\pi_{i_k}}\) of attributes. Let \(A_{\pi_1}, \ldots, A_{\pi_m}\) be the final ordering of all attributes. Then according to Definition 6.2, we construct a hierarchical DTD \(D'\) and set of FDs \(\Sigma'\) using this ordering. Now it suffices to show that \((D', \Sigma')\) is in XNF.

Suppose there is an FD \(X \rightarrow A_{\pi_i} \in \mathcal{F}^+\). This FD has a translation \(S \rightarrow p.@l_i\) in the semi-hierarchical representation \((D, \Sigma)\). Since \((D, \Sigma)\) is in XNF, the FD \(S \rightarrow p\) should also be in \((D, \Sigma)^+\). This means \(S\) implies all the elements and their attributes from the root to (and including) \(last(p)\). Therefore, \(X\) implies all attributes \(A_{\pi_j}, j < i\), since they all appear on the path from the root to \(last(p)\) in XML representation. Then, due to Theorem 6.3 and Corollary 6.4 the XML translation \((D', \Sigma')\) obtained from ordering \(A_{\pi_1}, \ldots, A_{\pi_m}\) is also in XNF. □
6.5 Related Work

XML documents are often views of relational data. Therefore, the problem of transforming data from the relational model to XML documents has been addressed in many papers (e.g., [CKS+00, FKS+02, LVL06, VLL04a, SSB+00]), some of which also consider translating constraints [LVL06, VLL04a]. Designing a redundancy-free XML document by taking FDs into account is addressed by Vincent et al. [VLL04a]. They show that any arbitrary mapping of a relation into a hierarchical XML document is redundancy-free if and only if the relation is in BCNF. However, the authors do not provide a characterization of cases when a redundancy-free XML document is obtainable from non-BCNF relational data, which is the main concern of this chapter.
Chapter 7

XML Design for Relational Storage

The design principles for XML data that we studied in Chapters 5 and 6 provide guidelines to eliminate redundancies and avoid update anomalies based on the assumption of a native XML storage. However, most XML data is stored in relational databases in practice.

In this chapter, we study XML design and normalization for relational storage of XML documents. To be able to evaluate the quality of a relational storage for XML, once more we use the information-theoretic framework [AL05], presented in Chapter 3, that measures information content in relations and documents, with higher values corresponding to lower levels of redundancy. We show that most common relational storage schemes preserve the notion of being well-designed (i.e., anomaly- and redundancy-free). Thus, existing XML normal forms guarantee well-designed relational storages as well.

We further show that if this perfect option is not achievable, then a slight restriction on XML constraints guarantees a reasonable amount of redundancy for the relational storage, achieving a “second-best” design. The results of this chapter were initially presented in [KL07].
Chapter 7. XML Design for Relational Storage

7.1 Introduction

With a recent shift to XML as a data model, many of classical database subjects have been reexamined in the XML context, among them design and normalization. The goal of normalization is to eliminate redundancies from a database or an XML document, and by doing so, eliminate or reduce potential update anomalies. With that goal in mind, XML normal forms have been introduced [AL04, WT05, VLL03] and proved to eliminate redundancies in data storage [AL05].

However, all of the works on XML design make an assumption of a native XML storage for reasoning about redundancies and anomalies. While native XML storage facilities exist [NdL05, LKA05, KM06, ACFR02], many XML processing systems take advantage of relational databases to store and query XML data [FK99, YASU01, SKWW01, TVB+02, DFS99]. The issues of storing and querying relational representations of XML have been studied extensively (refer to a survey by Krishnamurthy et al. [KKN03]). Now a natural question to ask is: do the existing principles of XML design apply when one stores XML in relational databases? And, in case there is a mismatch between the native XML representation and a relational one, how can one adjust XML design principles to guarantee well-designed relational representation of XML documents?

As we try to formulate these questions in a more precise way, one issue arises immediately: how do we compare XML designs and the designs of their relational translations? After all, the notions of redundancies, updates, queries, etc., are rather different in these two worlds. To overcome this problem, we use the information-theoretic approach, proposed by Arenas and Libkin [AL05] and reviewed in Chapter 3, that applies across different data formats and integrity constraints, and that allows us to reason about and compare data designs over different data models.

The idea of this approach is that it measures, in a way independent of features such as updates and queries, the information content of data, as an entropy of a suitably chosen probability distribution. The higher this information content is, the less redundancy the
design carries. Well-designed or perfect databases are the ones with the highest amount of information content.

When XML documents are stored as relations, constraints may not remain in the same forms, e.g., XML keys may change to functional dependencies over the relational schema [DFH07]. However, the information-theoretic approach applies to any kind of constraints, and as our first result, we are able to show that the normal form XNF corresponds precisely to the perfect designs of relational translations of XML. For this result, we impose fairly mild conditions on translations of XML; in fact for two popular translations used in the literature [STZ+99, FK99] the result holds.

While one often tries to achieve a perfect design, in practice it is not always possible. For example, if we deal with relational databases and functional dependencies, the perfect design that eliminates all redundancies is BCNF, but if one needs to enforce all the functional dependencies at the same time, a decomposition into BCNF may not exist [AHV95]. In that case, one usually tries to obtain a 3NF (3rd Normal Form) design; in fact, 3NF is much more commonly used in practice than BCNF. We showed in Chapter 4 that 3NF can be explained information-theoretically as the best relational normal form achievable if all functional dependencies are preserved. Using the information-theoretic measure, one can also characterize it as a normal form that always guarantees values of the measure of at least $\frac{1}{2}$, which is the highest value that one can guarantee if the preservation of functional dependencies is important.

In this chapter, we show that there is a simple XML design criterion that guarantees a second-best relational storage. Namely, our XML design criterion specifies that every constraint violating XNF should be relative [BDF+$01a$, AFL02a], i.e., restricted to some element type of a DTD that occurs under the scope of a Kleene star.

Thus, our results suggest the following guidelines for XML design if one stores XML documents in relations:

1. try to achieve the normal form XNF (using the normalization algorithm [AL04]);
2. if that fails, try replacing all XNF-violating constraints by relative ones.

This way one guarantees good design not only of an XML document itself, but also of its relational storage by removing redundancies.

Our final result deals with a relational representation of documents in which we essentially store the document as a tree (the edge relation) [FK99]. This representation shares some of the information-theoretic characteristics with those more commonly used, but we show that it may require arbitrarily many more relational joins for enforcing XML constraints even for well-designed documents in XNF.

The rest of this chapter is organized as follows. In Section 7.2 we overview relational translations of XML and XML constraints. In Section 7.3, we show that perfect XML designs (i.e., XNF) correspond precisely to perfect designs of relational representations. In Section 7.4 we find conditions on XML documents guaranteeing the best non-perfect relational designs, akin to 3NF. In Section 7.5 we discuss the edge representation and extend our results for this representation.

### 7.2 XML-to-Relational Mapping Scheme

The main goal of this chapter is to show how a good XML design can result in having a less redundant relational storage for XML documents. Redundancies occur when DTDs permit adding more redundant values, or redundant tree tuples, to XML documents. In particular, redundancies happen when element types occur under the scope of a Kleene star in the DTD. This is indeed what is required for the worst cases of redundancy, so for lower bounds of information content we can safely concentrate on DTDs that basically model nested relations, i.e., non-recursive disjunction-free DTDs, where each element type appears once or under a Kleene star in the production rule of its parent. We also assume that data elements come from a countably-infinite domain.

Different mapping schemes [STZ+99, FK99, YASU01, SKWW01, BFM05, DT03,
DFS99] have been proposed for transforming XML documents into relational data. We shall use a technique called inlining [STZ+99] as our main XML-to-relational mapping scheme. While the inlining technique is not the only or necessarily the best mapping scheme (see a comparison by Krishnamurthy et al. [KKN03]), it produces the most natural relational schema for DTDs that are essentially nested relations. But our results are more general: they apply to any other mapping scheme that produces a similar relational schema for the DTDs that we consider. We will also extend our results for another popular mapping scheme, called the edge representation.

Given a DTD, the basic idea of the inlining mapping is that separate relations are created for the root and the element types that appear under a star, and the other element types are inlined in the relations corresponding to their parents. Each relation corresponding to an element type has an ID attribute that is a key for that relation as well as a parent ID attribute that is a foreign key pointing to the parent of that element in the document. All the attributes of a given element type in the DTD become attributes in the relation corresponding to that element type.

**Example 7.1** Consider the XML tree of Figure 7.1 that conforms to DTD $D = (E, A, P, R, \text{db})$, where

$E = \{ \text{db, student, contact, address, phone} \}$

![Figure 7.1: An XML tree.](image-url)
\[ A = \{ \text{@name, @streetNo, @aptNo, @city, @postalCode, @number} \} \]

\[ P = \{ \text{db} \rightarrow \text{student}^*, \text{student} \rightarrow \text{contact}, \text{contact} \rightarrow \text{address}^*, \text{phone}^*, \text{address} \rightarrow \epsilon, \text{phone} \rightarrow \epsilon \} \]

\[ R(\text{student}) = \{ \text{@name} \} \]

\[ R(\text{address}) = \{ \text{@streetNo, @aptNo, @city, @postalCode} \} \]

\[ R(\text{phone}) = \{ \text{@number} \} \]

\[ R(\text{db}) = R(\text{contact}) = \emptyset \]

The relational database for this DTD that is produced using the inlining technique would have the following schema:

\[
\begin{align*}
\text{student}(\text{stID}, \text{name}, \text{conID}) \\
\text{address}(\text{addID}, \text{conID}, \text{postalCode}, \text{streetNo}, \text{aptNo}, \text{city}) \\
\text{phone}(\text{phID}, \text{conID}, \text{number})
\end{align*}
\]

Keys are underlined, and the following foreign key constraints hold among the attributes: \( \text{address[conID]} \subseteq_{FK} \text{student[conID]} \), \( \text{phone[conID]} \subseteq_{FK} \text{student[conID]} \).

The inlining schema generation can be formally captured as follows.

**Definition 7.2** Given a DTD \( D = (E, A, P, R, r) \), we define the **inlining** of \( D \) to be a relational database schema \( S \), where

- \( S = \{ R_e \mid e \in E, \text{ and } e^* \text{ occurs in } P_{e'} \text{ for some } e' \in E \text{ or } e \text{ occurs in } P_r \} \), and
- there is a mapping \( \sigma : E \rightarrow S \) recursively defined as

\[
\sigma(e) = \begin{cases} 
R_e & \text{if } R_e \in S \\
\sigma(e'), \text{ where } e \text{ occurs in } P_{e'} & \text{if } R_e \notin S
\end{cases}
\]

such that for each \( R_e \in S \),

\[
\text{sort}(R_e) = \bigcup_{e' \in \sigma^{-1}(R_e)} (\{e'ID\} \cup A(e')) \cup \{e'ID \mid e \text{ occurs in } P_{e'}\}.
\]

Then given an XML tree \( T = (V, \text{lab, ele, att, root}) \) conforming to \( D \) and satisfying \( \Sigma \), we can straightforwardly **shred** it into an instance \( I_T \) of relational schema \( S \), the inlining of \( D \). Note that we can use node identifiers in \( V \) for values of ID attributes when populating \( I_T \).


7.2.1 Translating XML Constraints

It is easy to observe that the FDs defined over a DTD do not necessarily translate into FDs over the relational schema, simply because the paths involved in an XML functional dependency may not all occur in a single relation. Therefore, we need to join different relations to enforce the integrity constraints that are now in the form of equality-generating dependencies (EGDs).

Example 7.3 Consider DTD $D$ in Example 7.1 and the XML tree in Figure 7.1 conforming to $D$. This tree satisfies the following constraints: (1) for each student, no more than one address is kept with a single postalCode, and (2) the value of postalCode uniquely determines streetNo and city. These constraints can be formulated as the following FDs:

$$
\text{db.student, db.student.contact.address.@postalCode} \rightarrow \text{db.student.contact.address} \quad (7.1)
$$

$$
\text{db.student.contact.address.@postalCode} \rightarrow \text{db.student.contact.address.@city} \quad (7.2)
$$

These FDs would translate into the following EGDs over the relational database schema that we obtained using the inlining technique. Note that EGD (7.4) is an FD, but EGD (7.3) is not.

$$
\forall \text{ student(s, n, c) } \land \text{ address(a, c, pc, st, apt, ct) } \land \\
\text{ student(s, n, c) } \land \text{ address(a', c, pc, st', apt', ct') } \rightarrow a = a', \quad (7.3)
$$

$$
\forall \text{ address(a, c, pc, st, apt, ct) } \land \\
\text{ address(a', c', pc, st', apt', ct') } \rightarrow ct = ct'. \quad (7.4)
$$

We can show that every constraint expressed in the form of an FD for XML can be written as an EGD over the inlining relational storage. More formally,
Proposition 7.4 For every DTD $D$ and set of XML functional dependencies $\Sigma$ defined over $D$, there is a set of EGDs $\Sigma_E$ over the relations of $S$, the inlining of $D$, such that for every XML tree $T$ conforming to $D$, the tree $T$ satisfies $\Sigma$ if and only if $I_T$ satisfies $\Sigma_E$.

Proof sketch: Recall that $\text{tuples}_D(T)$ is the set of maximal tree tuples in $T$. We can consider this set as a relation $R_T$ with a schema consisting of attribute names in $\text{paths}(D)$. Then every XML functional dependency is equivalent to a regular FD over the attributes of $R_T$. It can be shown that the inlining translation is a lossless representation of $R_T$, as the tree tuples and inlining models are both lossless representations of an XML tree. Then the proof of the proposition follows from the following lemma:

Lemma 7.5 Let $R$ be a relation schema and $S = \{R_1, \ldots, R_n\}$ be a lossless decomposition of $R$ and $X \rightarrow A$ be an FD over the attributes of $R$. Then there is an EGD $e$ over the relations of $S$ such that every instance $I$ of $R$ satisfies $X \rightarrow A$ if and only if $I_S$, the decomposition of $I$ into the relations of $S$, satisfies $e$. □

Note that there are also some key and foreign key constraints $\Sigma_{K,FK}$ over the ID attributes of $S$, as shown in the previous example. We then call $(S, \Sigma_S)$ the inlining translation of $(D, \Sigma)$, where $\Sigma_S = \Sigma_E \cup \Sigma_{K,FK}$.

7.3 Relational Storage for Perfect XML Designs

We know that if we want a perfectly non-redundant design for XML, we should try to achieve XNF [AL05]. XML is most commonly stored in relational databases, in order to take advantage of fast relational query engines and well-developed storage facilities. We therefore need to study the design and normalization not only for XML per se, but also for the relational storage of XML documents. Here we study the effect of XNF normalization on the relational storage, and in the next section we will see how the relational storage looks, in terms of redundancy, for non-perfect XML designs.
We observe that the inlining mapping preserves a good XML design by showing that the information content of data values will remain maximum after transforming XML data into relational storage. Note that the FDs over XML data will become EGDs over the relational storage.

Let \( D \) be a DTD, \( \Sigma \) be a set of XML functional dependencies defined over \( D \), and \((S, \Sigma_S)\) be the inlining translation of \((D, \Sigma)\). Consider an XML tree \( T \) conforming to \( D \) and satisfying \( \Sigma \). Let \( I_T \) be the instance of \((S, \Sigma_S)\) that is obtained by shredding \( T \) into relations of \( S \). Now every position \( p \) in \( T \) is naturally mapped into a unique position \( \delta(p) \) in \( I_T \). We now extend the definition of being well-designed, introduced in Chapter 3 and revisited in Chapter 5, for the inlining representation of an XML document.

**Definition 7.6** We say that an inlining translation \((S, \Sigma_S)\) of \((D, \Sigma)\) is well-designed if and only if for every XML tree \( T \) conforming to \( D \) and satisfying \( \Sigma \) and every position \( p \) in \( T \), we have \( \text{Ric}_{I_T}(\delta(p) \mid \Sigma_S) = 1 \).

In other words, in positions corresponding to the positions from an XML document, there is no redundancy whatsoever in the shredded document according to the translation of XML constraints.

**Theorem 7.7** The following are equivalent for an XML specification \((D, \Sigma)\) and its inlining translation \((S, \Sigma_S)\):

1. \((D, \Sigma)\) is well-designed (or, equivalently, is in XNF);

2. \((S, \Sigma_S)\) is well-designed.

**Proof sketch:** Let \( T \) be an XML tree conforming to \( D \) and satisfying \( \Sigma \), and \( I_T \) be an instance of \( S \) that is obtained by shredding \( T \) into the relations of the inlining translation. Let \( I_{T_{p=a}} \) and \( T_{p=a} \) denote an instance and a tree in which the value in position \( p \) is replaced by \( a \). Let \( \text{adom}(I_T) \) and \( \text{adom}(T) \) denote the active domain of
instance $I_T$ and tree $T$ respectively. Now the proof follows from the following statements:

For every tree $T$ conforming to $D$ and satisfying $\Sigma$ and arbitrary position $p \in Pos(T)$,

1. $\text{Ric}_{T}(p \mid \Sigma) = 1$ if and only if for a fresh value $a \not\in \text{adom}(T)$, $T_{p\leftarrow a}$ satisfies $\Sigma$ [AL05].

2. $\text{Ric}_{I_T}(\delta(p) \mid \Sigma_S) = 1$ if and only if for a fresh value $a \not\in \text{adom}(T)$, $I_{T_{p\leftarrow a}}$ satisfies $\Sigma_S$ [AL05].

3. For a fresh value $a \not\in \text{adom}(T)$, $T_{p\leftarrow a}$ satisfies $\Sigma$ if and only if $I_{T_{p\leftarrow a}}$ satisfies $\Sigma_S$ (by Proposition 7.4).

This means that in order to ensure a non-redundant relational storage for our XML data, we need to have an XNF design. In other words, XNF achieves nonredundant design no matter what type of storage – native or relational – is used.

### 7.4 What is an Almost-Perfect Design for XML?

Like in the relational case, bad XML designs may lead to very low information contents for positions in the relational storage. In fact, the information content of a position in a relational storage of an XML document can potentially be arbitrarily low.

**Proposition 7.8** For every $\varepsilon > 0$, we can find an XML design $(D, \Sigma)$ with inlining translation $(S, \Sigma_S)$ and an XML tree $T$ conforming to $D$ and satisfying $\Sigma$ with a position $p$, such that for the corresponding position $\delta(p)$ in $I_T$, we have $\text{Ric}_{I_T}(\delta(p) \mid \Sigma_S) < \varepsilon$.

**Proof:** Consider DTD $D = (E, A, P, R, r)$, where $E = \{r, A\}$, $A = \{@a_1, \ldots, @a_m\}$, $P(r) = A^*$, $P(A) = \epsilon$, $R(r) = \emptyset$, and $R(A) = \{@a_1, \ldots, @a_m\}$, and the following functional dependencies:

\[
\begin{align*}
    r.A.@a_2 & \rightarrow r.A.@a_1 \\
    \vdots \\
    r.A.@a_m & \rightarrow r.A.@a_1
\end{align*}
\]
The inlining representation of this DTD contains only one relation:

\[ R_A(AID, rID, a_1, \ldots, a_m), \]

with FDs of the form \( a_i \rightarrow a_1 \) for \( i \in [2, m] \). Now consider an instance of \( R_A \) that contains tuples of the form \((aid, rid, 1, \ldots, 1)\), where \( rid \) is a constant for the root identifier and \( aid \) ranges from 1 to \( t \) to distinguish \( t \) tuples. Consider position \( p = (R_A, 1, a_1) \) in the instance corresponding to attribute \( a_1 \) in the first tuple.

**Claim 7.9** The information content of position \( p \) is

\[
\text{RIC}_I(p \mid \Sigma) = \frac{1}{2^{m+t-1}} \sum_{i=0}^{m-1} \binom{m-1}{i} (1 + 2^{-i})^t.
\]

**Proof of Claim 7.9.** Omitted, refer to Claim 4.11 in Chapter 4.

The following shows that as long as \( t > \log_{4/3}(2/\varepsilon) \) and \( m > 1 + \log(2/\varepsilon) \), the information content of \( p \) is less than \( \varepsilon \):

\[
\text{RIC}_I(p \mid \Sigma) = \frac{1}{2^{m+t-1}} \sum_{i=0}^{m-1} \binom{m-1}{i} (1 + 2^{-i})^t
\]

\[
= \frac{1}{2^{m+t-1}} \left( 2^t + \sum_{i=1}^{m-1} \binom{m-1}{i} (1 + 2^{-i})^t \right)
\]

\[
< 2^{1-m} + \frac{1}{2^{m+t-1}} \sum_{i=0}^{m-1} \binom{m-1}{i} (1 + 2^{-i})^t
\]

\[
= 2^{1-m} + \frac{1}{2^{m+t-1}} \left( 2^{m-1}(\frac{3}{2})^t \right)
\]

\[
= 2^{1-m} + \left( \frac{3}{4} \right)^t
\]

\[
< \varepsilon.
\]

This completes the proof of Proposition 7.8. \( \square \)

To avoid the possibility of having such a high redundancy, we need some restrictions that guarantee a reasonable information content for all positions in the relational storage.
of XML. We showed in the previous section that an XNF design corresponds to maximum information content for the relational storage, but is it always possible to achieve XNF?

Recall that in the relational context it is not always possible to achieve a dependency-preserving perfect design. Therefore to guarantee dependency preservation, we may have to tolerate some redundancy, at most equal to one half of the maximum information content, and this is exactly what a good 3NF (i.e., 3NF+) normalization gives us.

The notion of dependency preservation is not well-understood for XML trees. However, producing an XNF design may not always be feasible for a set of functional dependencies, as several back-to-back XNF decompositions can excessively change the structure of a document and affect the performance of query answering.

The goal is therefore to find a second-best normal form that achieves a reasonable amount of redundancy without having to make drastic changes to the structure of the documents. Here we take a first step towards this goal by defining a simple restriction on FDs defined for XML that guarantees a good information content for relational storage of an XML document. The restriction is simply that all FDs should satisfy XNF or be relative to an element that occurs under the scope of a Kleene star. Then the information content of all positions of the relational storage of the XML document will be at least \( \frac{1}{2} \).

Let \( D = (E, A, P, R, r) \) be a DTD and \( \Sigma \) be a set of XML functional dependencies defined over \( D \).

**Definition 7.10** We say that an FD \( \{q_1, \ldots, q_n\} \rightarrow q \in \Sigma \) is relative under the Kleene star if

- \( q_i \) is an element path for some \( i \in [1, n] \),
- for all \( j \in [1, n] \), \( q_i \) is a prefix of \( q_j \), and
- for some \( p \) which is a prefix of \( q_i \) and \( \tau \in E \), \( \text{last}(p) \) occurs under a Kleene star in \( P_\tau \).
Example 7.11 Referring back to Examples 7.1 and 7.3, the functional dependency defined by expression (7.1) is an example of a relative FD, where the constraint holds within each fixed element student. The FD defined by expression (7.2) is not relative. We may refer to such FDs as absolute or global FDs.

7.4.1 Guaranteed Information Content

We now extend the definition of guaranteed information content, introduced in Chapter 4, for relational storage of XML documents. Let \( C \) be some condition on XML functional dependencies defined over a DTD, e.g., XNF or relative under the Kleene star. Now consider all XML trees \( T \) conforming to some DTD \( D \) and satisfying FDs \( \Sigma \), such that FDs in \( \Sigma \) are of type \( C \). The guaranteed information content of a condition \( C \) for relational mapping scheme \( s \) is the largest number \( c \in [0,1] \) such that for all such trees \( T \) and position \( p \) in \( T \), \( \text{Ric}_{I_T}(\delta(p) \mid \Sigma_S) \geq c \), where \( I_T \) is the shredding of \( T \) into the relations of schema \( S \) created by mapping scheme \( s \), \( \Sigma_S \) is the set of EGDs obtained from \( \Sigma \), and \( \delta(p) \) is the position in \( I_T \) to which \( p \) is mapped.

Definition 7.12 Let \( C \) be a condition on XML functional dependencies defined over a DTD. We define the set of possible values of \( \text{Ric}_{I_T}(\delta(p) \mid \Sigma_S) \) for XML trees \( T \) of XML designs satisfying \( C \):

\[
P\text{OSS}_C(s) = \{ \text{Ric}_{I_T}(\delta(p) \mid \Sigma_S) \mid T \text{ is an instance of } (D, \Sigma), S \text{ contains relations obtained from } D \text{ by mapping scheme } s, \Sigma_S \text{ contains EGDs obtained from } \Sigma, (D, \Sigma) \text{ satisfies } C \}\.
\]

Then guaranteed information content, \( XGIC_C(s) = \inf P\text{OSS}_C(s) \).

If \( \text{inl} \) denotes the inlining mapping scheme, we can reformulate Theorem 7.7, using the above definition, as \( XGIC_{XNF}(\text{inl}) = 1 \). Now let relative denote the condition of being
relative under a Kleene star. Then we can formally state the main result of this section:

**Theorem 7.13** $\text{XGIC}_{\text{relative}(\text{inl})} \geq \frac{1}{2}$.

In other words, if we manage to design an XML document in such a way that there is no global FDs, then the redundancy of each data value in the relational storage of the XML document would not be worse than $\frac{1}{2}$.

We can explain this result more intuitively by looking at update anomalies that could happen in the relational storage of an XML document. Most database management systems disallow updates that violate key constraints, but the only mechanism to enforce FDs or EGDs would be through writing assertions. In the absence of global or absolute FDs on the XML side, the possibility of FD or EGD violation due to a bad update in the relational storage will be restricted to a small portion of the entire database. Informally speaking, if we are to numerically evaluate the possibility of having update anomalies, our results state that by having global FDs, the relational storage of an XML document could be exponentially more prone to anomalies, compared to the case when we are restricted to XNF and relative FDs.

Before we prove Theorem 7.13 we need a lemma similar to Lemma 4.6 in Chapter 4.

Let $\Sigma$ be a set of EGDs over a database schema $S$, $I \in \text{inst}(S, \Sigma)$, $p \in \text{Pos}(I)$. We say that $\bar{a} \in \Omega(I, p)$ determines $p$ if there exists $k_0 > 0$ such that for every $k > k_0$, we have $P(a | \bar{a}) = 1$ for some $a \in \text{adom}(I)$, and $P(b | \bar{a}) = 0$ for every $b \in [1, k] - \{a\}$. In other words, $\bar{a}$ determines $p$ if one can specify a single value for $p$, given the values present in $\bar{a}$ and constraints $\Sigma$. We write $\Omega_0(I, p)$ for the set of all $\bar{a} \in \Omega(I, p)$ that determine $p$, and $\Omega_1(I, p)$ for the set of all $\bar{a} \in \Omega(I, p)$ that do not determine $p$. Let $n = |\text{Pos}(I)|$. Then:

**Lemma 7.14** $\text{RIC}_I(p | \Sigma) = |\Omega_1(I, p)|/2^{n-1}$.

We omit the proof of this lemma as it is exactly the same as that of Lemma 4.6. To prove Theorem 7.13, we need to show that for every XML specification $(D, \Sigma)$, where all
FDs in $\Sigma$ are relative or satisfy XNF, every XML tree $T$ conforming to $D$ and satisfying $\Sigma$, and every position $p \in \text{Pos}(T)$, we have $\text{Ric}_{I_T}(\delta(p) \mid \Sigma_S) \geq 1/2$. Consider an arbitrary instance $T$ of such an XML specification that is shredded into the relations of inlining schema $S$, with EGDs $\Sigma_S$. Let $p' = (R_e, t_0, A)$ be the position, to which a position $p = (x, @a)$ in $\text{Pos}(T)$ is mapped ($p' = \delta(p)$). According to the inlining mapping scheme, relation $R_e$ has two attributes $eID$ and $e'ID$ corresponding to node identifiers of the nodes labeled $e$ and their parents, respectively. Let $p_e$ denote the path in $\text{paths}(D)$ whose last element type is $e$ and $p. @a$ denote the attribute path in $\text{paths}(D)$ connecting the root to the position $p = (x, @a)$. We need an easy observation that for a relative FD $\{q_1, \ldots, q_n\} \rightarrow q \in (D, \Sigma)^+$, where $p_e$ is a prefix of all paths $q_1, \ldots, q_n$, the FD $\{q_1, \ldots, q_n\} \rightarrow p. @a$ is trivial. Then we can conclude that every nontrivial FD of the form $X \rightarrow p. @a \in (D, \Sigma)^+$ either satisfies XNF or is relative with respect to a path that is a prefix of $p_e$.

Now let $\bar{a}$ be a vector in $\Omega(I_T, p')$ and $\bar{a}_{[t_0]}$ denote the subtuple of $\bar{a}$ corresponding to $t_0$. If $\bar{a}_{[t_0]}$ contains a variable in the position corresponding to attribute $e'ID$, the connection of the subtree rooted at the node labeled $e$ that contains position $p$ to its parents will be completely lost. Then given that all non-XNF FDs in $(D, \Sigma)^+$ implying $p. @a$ are relative with respect to paths that are prefixes of $p_e$, the corresponding EGDs will not impose a value for $p'$, and hence such $\bar{a}$ does not determine $p'$ no matter what the other positions in $\bar{a}$ contain. All other $n-2$ positions in $\bar{a}$ can therefore contain either a constant or a variable, so there are at least $2^{n-2}$ $\bar{a}$’s that do not determine $p'$. Then using Lemma 7.14, we conclude that the information content of $p'$ is at least $\frac{2^{n-2}}{2^{n-1}} = 1/2$. This proves Theorem 7.13. $\blacksquare$
7.5 A Different Mapping Scheme

The goal of this section is to show that the XML design criteria that we obtained to have a good relational storage is not limited to only one mapping scheme. Beside inlining, there are other XML-to-relational data and query mapping schemes [FK99, YASU01, SKWW01, BFM05, DT03, DFS99], among them being the Edge representation [FK99], which is used as a basis for many XML query translation techniques. Here we would like to study this mapping scheme from two points of view: 1) redundancy and information content, and 2) the complexity of enforcing integrity constraints.

In the Edge representation, an XML tree is viewed as an edge-labeled graph. Each element-to-element and element-to-attribute edge of the tree has a tuple in Edge table, and for each data value in the tree, there is a tuple in Value table associating a node identifier to a value. The relational storage for any XML tree, regardless of its schema, has the following relations:

\[
\text{Edge}(source, target, label),
\]

\[
\text{Value}(vid, val).
\]

In the original definition of this schema [FK99], there are two more attributes in the Edge relation: ordinal, which specifies the ordinal of the target among children of the source, and flag which specifies whether the tuple corresponds to an element-to-element edge or an element-to-attribute edge. We simplify the schema by removing these attributes, as they do not have any effect on the redundancy of data values.

Given an XML tree \(T = (V, lab, ele, att, root)\), we can populate Edge and Value by shredding \(T\) in the following way:

- for each pair of nodes \(x, y \in V\) such that \(y\) occurs in \(ele(x)\), there is a tuple \((x, y, lab(y)) \in Edge\), and

- for each \(x \in V\) and attribute \(@a\) such that \(att(x, @a)\) is defined, there is a tuple \((x, y, @a) \in Edge\) as well as a tuple \((y, att(x, @a)) \in Value\), where \(y \notin V\) is a fresh
Other variants of this approach also exist. One of them is the binary approach [FK99, SKWW01], where the Edge relation is horizontally partitioned based on attribute label. The schema would then have the following relations:

\[ B_{\text{label}}(\text{source}, \text{target}) \]
\[ \text{Value}(\text{vid}, \text{val}) \]

In this representation, edge labels or element types are not stored as attributes. We can therefore assume that the domain of each attribute is an infinite set, and thus the measure of information content can be directly applied.

Given a set of FDs \( \Sigma \) defined over a DTD \( D \), the translation of \( \Sigma \) over the binary representation of \( D \) will again be a set of EGDs. We denote the binary translation of \( (D, \Sigma) \) by \( (S_B, \Sigma_B) \), where \( \Sigma_B \) contains some key and foreign key constraints, as well as some EGDs obtained from \( \Sigma \). Every \( T \) conforming to \( D \) and satisfying \( \Sigma \) can be trivially shredded into an instance \( I_B \) of \( (S_B, \Sigma_B) \), and each position \( p \) in \( T \) is mapped to a unique position \( \delta_B(p) \) in \( I_B \).

We now focus on the information content of binary representation and show that, redundancy-wise, binary and inlining representations look the same. The definitions of being well-designed for a binary representation of an XML specification is very similar to that of the inlining translation.

**Definition 7.15** We say that a binary translation \( (S_B, \Sigma_B) \) of \( (D, \Sigma) \) is well-designed if and only if for every XML tree \( T \) conforming to \( D \) and satisfying \( \Sigma \) and every position \( p \) in \( T \), we have \( \text{Ric}_{I_B}(\delta_B(p) \mid \Sigma_B) = 1 \).

Not surprisingly, the binary translation also preserves a perfect XML design:

**Theorem 7.16** The following are equivalent for an XML specification \( (D, \Sigma) \) and its binary translation \( (S_B, \Sigma_B) \):
1. \((D, \Sigma)\) is well-designed (or, equivalently, is in XNF);

2. \((S_B, \Sigma_B)\) is well-designed.

Recall that \(XGIC_{\text{relative}}(\text{bin})\) denotes the guaranteed information content for all positions in a binary representation of an XML document when all FDs either satisfy XNF or the relative under star condition. The following theorem shows that, similar to the inlining representation, to guarantee an information content of \(\frac{1}{2}\) for positions in the binary representation of an XML document, it is enough to make sure that all the XML functional dependencies are relative:

**Theorem 7.17** \(XGIC_{\text{relative}}(\text{bin}) \geq \frac{1}{2}\).

This might make us think that, from the design point of view, binary and inlining representations are equivalent. However, we will next show that they differ significantly when it comes to the complexity of enforcing integrity constraints. Here by complexity, we mean the number of joins needed to write a SQL assertion that enforces an EGD translated from an XML functional dependency. Given an XML functional dependency \(f\) over a DTD \(D\), we use the notation \(\# \land\lor f\)\_\text{inl}\) and \(\# \land\lor f\)\_\text{bin}\) to denote the number of joins required to enforce the EGD resulting from the translation of \(f\) on the inlining and binary relational representation of \(D\) respectively.

Consider for instance FD (7.2) of Example 7.3. The translation of this FD over the binary representation of the DTD is an EGD that involves five joins (\(\# \land\lor f\)\_\text{bin} = 5), whereas over the inlining translation, it can be written as an FD requiring only one join (\(\# \land\lor f\)\_\text{inl} = 1), as shown by EGD (7.4). In fact, we can show that even for XNF dependencies, the number of joins needed for the binary representation is never smaller than under inlining and can be arbitrarily higher.

**Proposition 7.18**

1. For every DTD \(D\) and XML functional dependency \(f\), we have \(\# \land\lor f\)\_\text{bin} \geq \# \land\lor f\)\_\text{inl}. 
2. For every \( m > 0 \), we can find a DTD \( D \) and an XML functional dependency \( f \) over \( D \), such that

\[
\frac{\# \text{bin}_f}{\# \text{inl}_f} > m.
\]

Proof: The first statement is trivial if we notice that the binary representation is nothing but a parent-child relation, while the inlining representation provides access to the attributes of a node as well as some of its non-child descendants without requiring a join.

To observe that the second statement is true, consider DTD \( D = (E, A, P, R, r) \), where \( E = \{r, A\} \), \( A = \{a_1, \ldots, a_k\} \), \( P(r) = A^* \), \( P(A) = \epsilon \), \( R(r) = \emptyset \), and \( R(A) = \{a_1, \ldots, a_k\} \), and the following functional dependency which satisfies XNF:

\[
r.A.a_1, \ldots, r.A.a_k \rightarrow r.A
\]  

(7.5)

The inlining representation of this DTD contains only one relation:

\[
R_A(AI, rID, a_1, \ldots, a_k),
\]

and FD 7.5 can be translated into the following EGD with only one self join:

\[
\forall R_A(aid, rid, a_1, \ldots, a_k) \land \\
R_A(aid', rid, a_1, \ldots, a_k) \rightarrow aid = aid'.
\]  

(7.6)

The EGD corresponding to FD (7.5) over the binary schema would look like the following expression:

\[
\forall B_{a_1}(s_1, t_1^1) \land Value(t_1^1, a_1) \land \\
\vdots \\
B_{a_k}(s_1, t_k^1) \land Value(t_k^1, a_k) \land \\
B_{a_1}(s_2, t_1^2) \land Value(t_1^2, a_1) \land \\
\vdots \\
B_{a_k}(s_2, t_k^2) \land Value(t_k^2, a_k) \rightarrow s_1 = s_2,
\]  

(7.7)

which consists of \( 2k - 1 \) joins. By taking \( k > (m + 1)/2 \) we can have \( \frac{\# \text{bin}_f}{\# \text{inl}_f} > m \), which proves the proposition. \( \square \)
Related Work

Several papers have addressed the problem of storing XML data in relational databases and constraint transformation [LC01, DFH07, CDHZ03]. Designing a relational schema for an XML design that reducing redundancy was addressed by Chen et al. [CDHZ03]. They introduce an algorithm that produces a 3NF relational schema for a given DTD and a set of XML functional dependencies. Their work is different from what we considered in this chapter in the following ways: first, they produce a relational representation of XML that only contains data values, and the structural information (e.g., parent-child relationships between the XML tree nodes) is not stored. Furthermore, XML functional dependencies always translate into relational keys and functional dependencies.

Davidson et al. [DFH07] show how a good (e.g., BCNF or 3NF) relational storage can be designed for XML data by considering the problem of propagating relational functional dependencies from a set of XML key dependencies. This work does not consider mappings that preserve the structure of documents, and only XML keys are taken into consideration. A polynomial time algorithm is presented that checks whether an FD on a predefined relational schema is propagated from a set of XML keys. Another algorithm is presented that finds a minimal cover for all FDs mapped from a set of XML keys.

Lee et al. [LC01] consider a broader class of constraints for XML, including cardinality constraints, inclusion dependencies, and equality-generating dependencies, and show how the inlining translation algorithm can be modified, so that the transformation is constraint-preserving. This work, however, does not address the problem of reducing redundancy in the relational storage of an XML document. In fact, none of the papers mentioned in this section address the XML design problem to improve the redundancy of the relational storage, which is the main concern of this chapter.
Chapter 8

Conclusions

In this dissertation, we tried to make one step beyond the classical database design theory and propose new design guidelines that help a database carry less redundancy. Our criterion for a good design was based on how much redundant data the database can potentially store. We therefore defined a measure of highest redundancy, or equivalently lowest information content, based on the information-theoretic framework of Arenas and Libkin [AL05]. We then looked for the characteristics of database designs for which the amount of redundancy is the lowest possible or at least reasonable. Here we summarize our results that could be taken into account, when designing a relational or XML database, to reduce or eliminate the potential for redundancy.

Our first result concerns the design of relational databases with functional dependencies. We showed that when preserving functional dependencies is critical, a minimum amount of redundancy must be tolerated. To achieve this minimum redundancy, one has to normalize the database into a 3NF$^+$ design, which is the name we used for good 3NF designs produced by the main 3NF synthesis algorithm. Doing an arbitrary 3NF normalization can also reduce the redundancy, however, by a small factor.

Not every non-3NF$^+$ design is bad, and normalizing a database which does not carry much redundancy is a poor design decision that leads to inefficient query answering.
However, it is not immediately obvious that how much potential an arbitrary relational schema has for storing redundant data. Given a schema and a set of functional dependencies, we introduced a way of calculating the lowest possible information content for the instances of that schema to see where in the spectrum of redundancy the schema lies. Then it would be up to the database designer to decide if a normalization is necessary.

Our next main result addresses the design of XML documents that are views of relational data with functional dependencies. The redundancy of such a document highly depends on how the hierarchical structure of the document is designed. We characterized relational schemas, for which we can have a dependency-preserving XML design that completely eliminates redundancies, and proposed an algorithm that produces such an XML design for relational schemas that have the necessary properties. This way we can have a dependency-preserving redundancy-free representation for some of the relations for which such a normalization is not possible in the relational context.

Finally, we studied the design and normalization for XML documents stored in relations, rather than in native XML storage facilities. We once more used the information-theoretic tool to reason about the redundancy of such a storage. The conclusions are as follows: the XML normal form XNF, proposed [AL04] as a generalization of BCNF for XML documents, achieves the best possible design from the point of view of eliminating redundancies in both native and relational storage of XML. Note that algorithms for converting XML designs into XNF exist. If XNF is not possible, a reasonable redundancy can be achieved by making sure that all non-XNF functional dependencies are relative with respect to the scope of an element that occurs in a DTD with a Kleene star.

The information-theoretic framework also allows us to compare different shredding techniques from the point of view of redundancies in XML documents. We showed that two popular techniques – inlining [STZ+99] and the edge representation [FK99] – behave in the same way information-theoretically, while the latter can behave arbitrarily worse in terms of enforcing integrity constraints.
Chapter 9

Future Work

In this dissertation, we proposed a way to evaluate schemas based on the worst cases of redundancy, or the lowest cases of information content. Our motivation was that the more redundancy a schema allows, the more instances are prone to update and insertion anomalies. It would be interesting to find the connection between this measure and the quality of schemas in practice. For instance, we can expose a heavily-populated instance of a schema with functional dependencies to a large number of updates and insertions and measure the inconsistency of the resulting database. We can study how this inconsistency relates to the lowest information content that the schemas allow. We expect that the inconsistency is worse for schemas with a low guaranteed information content. This would be a nice experimental justification for our measure of comparison for schemas.

We also found a restrictive condition for XML functional dependencies that guarantees a reasonably-low redundancy for relational storage. It would be interesting to find out whether there is a normalization for every XML design that does not satisfy this condition. If not, there might be a less-restrictive normal form that guarantees a good information content, and every XML document admits a normalization into that normal form. Then this normal form can be the equivalent of 3NF for XML data.

Finally, it would be interesting to make further attempts to reexamine some of the
classical concepts in relational database design that are still open in the XML world. For instance, we do not yet have an adequate understanding of the notion of dependency preservation and its impact on XML design. The complicated structure of XML documents makes it nontrivial to define such a concept. It would be interesting to start with finding a precise way of checking whether a certain constraint (e.g., key or functional dependency) on data values is expressible over a given XML design.
Bibliography


[CKS+00] Michael J. Carey, Jerry Kiernan, Jayavel Shanmugasundaram, Eugene J. Shekita, and Subbu N. Subramanian. XPERANTO: Middleware for pub-


