THE THEORY OF RELATIONAL DATABASES

THE THEORY OF RELATIONAL DATABASES

DAVID MAIER
OREGON GRADUATE CENTER

π PITMAN

PITMAN PUBLISHING LIMITED
128 Long Acre, London WC2E 9AN

Associated Companies
Pitman Publishing New Zealand Ltd, Wellington
Pitman Publishing Pty Ltd, Melbourne

First published in Great Britain 1983
First published in USA 1983

This book was first published in 1983 by
Computer Science Press, Inc.
11 Taft Ct.
Rockville, Maryland 20850

Printing 1 2 3 4 5 87 86 85 84 83 Year

Library of Congress Cataloging in Publication Data

Maier, David, 1953-
The theory of relational databases.

(Computer software engineering series)
Bibliography: p.
Includes index.
1. Data base management. I. Title. II. Series.
QA76.9.D3M33 001.64 82-2518
ISBN 0-914894-42-0 AACR2
UK ISBN 273-086-227

PREFACE

This book is a revision and extension of notes I wrote for a graduate seminar in relational database theory given at Stony Brook. The purpose of that course was to give students enough background in relational database theory to enable them to understand the current research being done in the field. I have not attempted to be exhaustive in covering all results in relational database theory—the field has already grown too large to cover everything. Instead, I have attempted to get within "one paper" of all current work: This book should give a student sufficient background to read recent papers in relational theory.

While most of the material presented here has been presented before, there is some new material, particularly on annular covers and in the chapter on database semantics. I have tried to bring material together that was available previously only in separate papers, and give some coherence to the results. That task has involved translating many of the results into standard notation, redoing some of the definitions, and constructing some new proofs for previously known theorems.

The book is aimed at a second course in databases, presumably at the graduate level, but possibly at the advanced undergraduate level. While an introductory course in database management systems is not an absolute prerequisite for this book, it is certainly desirable for some concrete motivation and intuition for the abstractions presented here. No specific course in mathematics is assumed, but there should be an acquaintance with set theory and the rudiments of formal logic. Some of the exercises require some sophisticated combinatorics, but those exercises are not central to the topic being developed—they are included for fun. Exercises that are deemed particularly difficult are marked with an asterisk.

Of course, I hope the book also will be a useful reference for researchers already working in the area. The bibliography is current through October 1981; some of the technical reports presumably have since appeared in journals and conference proceedings. I am grateful to Jeff Ullman for an advance copy of the bibliography to the second edition of *Principles of Database Systems*.

TABLE OF CONTENTS

Preface xiv
Acknowledgements 638

1. RELATIONS AND RELATION SCHEMES 1
1.1. Brass Tacks 1
1.2. Formalization of Relations 2
1.3. Keys 4
1.4. Updates to Relations 5
1.5. Exercises 8
1.6. Bibliography and Comments 10

2. RELATIONAL OPERATORS 11
2.1. Boolean Operations 11
2.2. The Select Operator 13
2.3. The Project Operator 15
2.4. The Join Operator 16
2.5. Properties of Join 18
2.6. Exercises 22
2.7. Bibliography and Comments 24

3. MORE OPERATIONS ON RELATIONS 25
3.1. The Divide Operator 25
3.2. Constant Relations 26
3.3. Renaming Attributes 27
3.4. The Equijoin Operator 29
3.5. Extensions for Other Comparisons on Domains 31
3.5.1. Extending Selection 32
3.5.2. The Theta-Join Operator 33
3.6. Relational Algebra 34
3.6.1. Algebraic Expressions as Mappings 35
3.6.2. Restricting the Set of Operators 36

3.7. The Split Operator 37
3.8. The Factor Operator 38
3.9. Exercises 39
3.10. Bibliography and Comments 41

4. FUNCTIONAL DEPENDENCIES 42
4.1. Definitions 42
4.2. Inference Axioms 44
4.3. Applying the Inference Axioms 47
4.4. Completeness of the Inference Axioms 49
4.5. Derivations and Derivation DAGs 51
4.5.1. RAP-Derivation Sequences 53
4.5.2. Derivation DAGs 56
4.5.3. More about Derivation DAGs 60
4.6. Testing Membership in F^+ 63
4.7. Exercises 69
4.8. Bibliography and Comments 70

5. COVERS FOR FUNCTIONAL DEPENDENCIES 71
5.1. Covers and Equivalence 71
5.2. Nonredundant Covers 72
5.3. Extraneous Attributes 74
5.4. Canonical Covers 77
5.5. The Structure of Nonredundant Covers 78
5.6. Minimum Covers 79
5.6.1. Direct Determination 79
5.6.2. Computing Minimum Covers 84
5.7. Optimal Covers 86
5.8. Annular Covers and Compound Functional Dependencies 87
5.9. Exercises 90
5.10. Bibliography and Comments 92

6. DATABASES AND NORMAL FORMS 93
6.1. Databases and Database Schemes 94
6.2. Normal Forms for Databases 96
6.2.1. First Normal Form 96
6.2.2. Anomalies and Data Redundancy 98
6.2.3. Second Normal Form 99
6.2.4. Third Normal Form 99
6.3. Normalization through Decomposition 101

6.4. Shortcomings of Normalization through Decomposition ... 104
6.5. Normalization through Synthesis ... 107
6.5.1. Preliminary Results for the Synthesis Algorithm ... 108
6.5.2. Developing the Synthesis Algorithm ... 108
6.5.3. Correctness and Other Properties of the Synthesis Algorithm ... 110
6.5.4. Refinements of the Synthesis Algorithm ... 113
6.6. Avoidable Attributes ... 115
6.7. Boyce-Codd Normal Form ... 117
6.7.1. Problems with Boyce-Codd Normal Form ... 119
6.8. Exercises ... 119
6.9. Bibliography and Comments ... 122

7. MULTIVALUED DEPENDENCIES, JOIN DEPENDENCIES, AND FURTHER NORMAL FORMS ... 123
7.1. Multivalued Dependencies ... 124
7.2. Properties of Multivalued Dependencies ... 126
7.3. Multivalued Dependencies and Functional Dependencies ... 127
7.4. Inference Axioms for Multivalued Dependencies ... 129
7.4.1. Multivalued Dependencies Alone ... 129
7.4.2. Functional and Multivalued Dependencies ... 132
7.4.3. Completeness of the Axioms and Computing Implications ... 133
7.5. Fourth Normal Form ... 135
7.6. Fourth Normal Form and Enforceability of Dependencies ... 137
7.7. Join Dependencies ... 139
7.8. Project-Join Normal Form ... 140
7.9. Embedded Join Dependencies ... 142
7.10. Exercises ... 143
7.11. Bibliography and Comments ... 144

8. PROJECT-JOIN MAPPINGS, TABLEAUX, AND THE CHASE ... 146
8.1. Project-Join Mappings ... 146
8.2. Tableaux ... 148
8.2.1. Tableaux as Mappings ... 150
8.2.2. Representing Project-Join Mappings as Tableaux ... 151
8.3. Tableaux Equivalence and Scheme Equivalence ... 152

8.4. Containment Mappings 156
8.5. Equivalence with Constraints 160
8.5.1. F-rules 162
8.5.2. J-rules 163
8.6. The Chase 164
8.6.1. The Finite Church-Rosser Property 168
8.6.2. Equivalence of Tableaux under Constraints 174
8.6.3. Testing Implication of Join Dependencies 175
8.6.4. Testing Implication of Functional Dependencies .. 177
8.6.5. Computing a Dependency Basis 180
8.7. Tableaux as Templates 182
8.8. Computational Properties of the Chase Computation 186
8.9. Exercises 189
8.10. Bibliography and Comments 194

9. REPRESENTATION THEORY 195
9.1. Notions of Adequate Representation 195
9.2. Data-Equivalence of Database Schemes 208
9.3. Testing Adequate Representation and Equivalence Under Constraints 210
9.3.1. **P** Specified by Functional Dependencies Only 211
9.3.2. **P** Specified by Functional and Multivalued Dependencies 215
9.3.3. Testing Data-Equivalence 217
9.4. Exercises 221
9.5. Bibliography and Comments 223

10. QUERY SYSTEMS 224
10.1. Equivalence and Completeness 225
10.2. Tuple Relational Calculus 227
10.2.1. Tuple Calculus Formulas 229
10.2.2. Types, and Free and Bound Occurrences 231
10.2.3. Tuple Calculus Expressions 236
10.3. Reducing Relational Algebra with Complement to Tuple Relational Calculus 242
10.4. Limited Interpretation of Tuple Calculus Formulas 244
10.4.1. Reducing Relational Algebra to Tuple Calculus with Limited Evaluation 247
10.4.2 Safe Tuple Calculus Expressions 247
10.5. Domain Relational Calculus 250
10.6. Reduction of Tuple Calculus to Domain Calculus 255

10.7. Reduction of Domain Calculus to Relational Algebra 257
10.8. Tableau Queries 262
10.8.1. Single Relation Tableau Queries 262
10.8.2. Tableau Queries for Restricted Algebraic Expressions 268
10.8.3. Tableau Queries that Come from Algebraic Expressions 272
10.8.4. Tableau Queries for Multirelation Databases 274
10.8.5. Tableau Set Queries 276
10.9. Conjunctive Queries 278
10.10. Exercises 278
10.11. Bibliography and Comments 286

11. QUERY MODIFICATION 287
11.1. Levels of Information in Query Modification 293
11.2. Simplifications and Common Subexpressions in Algebraic Expressions 295
11.3. Optimizing Algebraic Expressions 301
11.4. Query Decomposition 307
11.4.1. Instantiation 311
11.4.2. Iteration 313
11.4.3. The Query Decomposition Algorithm 315
11.5. Tableau Query Optimization 323
11.5.1. Tableau Query Equivalence 323
11.5.2. Simple Tableau Queries 327
11.5.3. Equivalence with Constraints 335
11.5.4. Extensions for Multiple-Relation Databases 339
11.5.5. Tableau Set Query Equivalence 348
11.6. Optimizing Conjunctive Queries 350
11.7. Query Modification for Distributed Databases 353
11.7.1. Semijoins 354
11.7.2. Fragments of Relations 359
11.8. Exercises 361
11.9. Bibliography and Index 369

12. NULL VALUES, PARTIAL INFORMATION AND DATABASES SEMANTICS 371
12.1. Nulls 372
12.2. Functional Dependencies and Nulls 377
12.3. Constraints on Nulls 384
12.4. Relational Algebra and Partial Relations 386

12.4.1. Possibility Functions 386
12.4.2. Generalizing the Relational Operators 389
12.4.3. Specific Possibility Functions 394
12.5. Partial Information and Database Semantics 406
12.5.1. Universal Relation Assumptions 406
12.5.2. Placeholders and Subscheme Relations 408
12.5.3. Database Semantics and Window Functions 410
12.5.4. A Window Function Based on Joins 413
12.5.5. Weak Instances 416
12.5.6. Independence 422
12.5.7. A Further Condition on Window Functions 427
12.6. Exercises 432
12.7. Bibliography and Comments 437

13. ACYCYLIC DATABASE SCHEMES 439
13.1. Properties of Database Schemes 439
13.1.1. Existence of a Full Reducer 439
13.1.2. Equivalence of a Join Dependency to Multivalued Dependencies 442
13.1.3. Unique 4NF Decomposition 443
13.1.4. Pairwise Consistency Implies Total Consistency 444
13.1.5. Small Intermediate Joins 445
13.2. Syntactic Conditions on Database Schemes 447
13.2.1. Acyclic Hypergraphs 447
13.2.2. Join Trees 452
13.2.3. The Running Intersection Property 455
13.3. Equivalence of Conditions 455
13.3.1. Graham Reduction 456
13.3.2. Finding Join Trees 457
13.3.3. The Equivalence Theorem for Acyclic Database Schemes 460
13.3.4. Conclusions 477
13.4. Exercises 478
13.5. Bibliography and Comments 482

14. ASSORTED TOPICS 485
14.1. Logic and Data Dependencies 485
14.1.1. The World of Two-Tuple Relations 486
14.1.2. Equivalence of Implication for Logic and Functional Dependencies 488

14.1.3. Adding Multivalued Dependencies 489
14.1.4. Nonextendability of Results 492
14.2. More Data Dependencies 493
14.2.1. Template Dependencies 494
14.2.2. Examples and Counterexamples for Template Dependencies 498
14.2.3. A Graphical Representation for Template Dependencies 500
14.2.4. Testing Implication of Template Dependencies 506
14.2.5. Generalized Functional Dependencies 516
14.2.6. Closure of Satisfaction Classes Under Projection 524
14.3. Limitations of Relational Algebra 527
14.4. Computed Relations 533
14.4.1. An Example 533
14.4.2. Testing Expressions Containing Computed Relations 536
14.5. Exercises 542
14.6. Bibliography and Comments 547

15. RELATIONAL QUERY LANGUAGES 550
15.1. ISBL 551
15.2. QUEL 556
15.3. SQL 561
15.4. QBE 568
15.5. PIQUE 583
15.6. Bibliography and Comments 591

BIBLIOGRAPHY 593

INDEX 611

Chapter 1

RELATIONS AND RELATION SCHEMES

One of the major advantages of the relational model is its uniformity. All data is viewed as being stored in tables, with each row in the table having the same format. Each row in the table summarizes some object or relationship in the real world. Whether the corresponding entities in the real world actually possess the uniformity the relational model ascribes to them is a question that the user of the model must answer. It is a question of the suitability of the model for the application at hand.

Whether or not the relational model is appropriate for a particular set of data shall not concern us. There are plenty of instances where the model is appropriate, and we always assume we are dealing with such instances.

1.1 BRASS TACKS

So much for philosophy. Let us consider an example. An airline schedule certainly exhibits regularity. Every flight listed has certain characteristics. It is a flight from an origin to a destination. It is scheduled to depart at a specific time and arrive at a later time. It has a flight number. Part of an airline schedule might appear as in Table 1.1.

What do we observe about this schedule? Each flight is summarized as a set of values, one in each column. There are restrictions on what information may appear in a given column. The FROM column contains names of airports served by the airline, the ARRIVES column contains times of day. The order of the columns is immaterial as far as information content is concerned. The DEPARTS and ARRIVES columns could be interchanged with no change in meaning. Finally, since each flight has a unique number, no flight is represented by more than one row.

The schedule in Table 1.1 is an example of a relation of type FLIGHTS. The format of the relation is determined by the set of column labels {NUMBER, FROM, TO, DEPARTS, ARRIVES}. These column names are

Table 1.1 FLIGHTS (airline schedule).

NUMBER	FROM	TO	DEPARTS	ARRIVES
83	JFK	O'Hare	11:30a	1:43p
84	O'Hare	JFK	3:00p	5:55p
109	JFK	Los Angeles	9:50p	2:52a
213	JFK	Boston	11:43a	12:45p
214	Boston	JFK	2:20p	3:12p

called attribute names. Corresponding to each attribute name is a set of permissible values for the associated column. This set is called the domain of the attribute name. The domain of NUMBER could be the set of all one-, two- or three-digit decimal integers. Each row in the relation is a set of values, one from the domain of each attribute name. The rows of this relation are called 5-tuples, or tuples in general. The tuples of a relation form a set, hence there are no duplicate rows. Finally, there is a subset of the attribute names with the property that tuples can be distinguished by looking only at values corresponding to attribute names in the subset. Such a subset is called a key for the relation. For the relation in Table 1.1, {NUMBER} is a key.

1.2 FORMALIZATION OF RELATIONS

We now formalize the definitions of the last section and add a couple of new ones. A *relation scheme* R is a finite set of *attribute names* $\{A_1, A_2, \ldots, A_n\}$. Corresponding to each attribute name A_i is a set D_i, $1 \leq i \leq n$, called the *domain* of A_i. We also denote the domain of A_i by $dom(A_i)$. Attribute names are sometimes called *attribute symbols* or simply *attributes*, particularly in the abstract. The domains are arbitrary, non-empty sets, finite or countably infinite. Let $\mathbf{D} = D_1 \cup D_2 \cup \cdots \cup D_n$. A *relation* r on relation scheme R is a finite set of mappings $\{t_1, t_2, \ldots, t_p\}$ from R to $\mathbf{D}$ with the restriction that for each mapping $t \in r$, $t(A_i)$ must be in D_i, $1 \leq i \leq n$. The mappings are called *tuples*.

Example 1.1 In Table 1.1 the relation scheme is FLIGHTS = {NUMBER, FROM, TO, DEPARTS, ARRIVES}. The domains for each attribute name might be:

1. dom(NUMBER) = the set of one-, two- or three-digit decimal numbers,

2. *dom*(FROM) = *dom*(TO) = {JFK, O'Hare, Los Angeles, Boston, Atlanta},
3. *dom*(DEPARTS) = *dom*(ARRIVES) = the set of times of day.

The relation in Table 1.1 has five tuples. One of them is t defined as t(NUMBER) = 84, t(FROM) = O'Hare, t(TO) = JFK, t(DEPARTS) = 3:00p, t(ARRIVES) = 5:55p.

Where did the mappings come from? What happened to tables and rows? We use mappings in our formalism to avoid any explicit ordering of the attribute names in the relation scheme. As we noted in the last section, such an ordering adds nothing to the information content of a relation. We do not want to restrict tuples to be sequences of values in a certain order. Rather, a tuple is a set of values, one for each attribute name in the relation scheme.* The mappings we defined are nothing more than correspondences of this type. Now that we have taken the trouble of avoiding any explicit ordering in relations, in nearly every case we shall denote our relations by writing the attributes in a certain order and the tuples as lists of values in the same order.

In either case, it makes sense, given a tuple t, to discuss the *value* of t on attribute A, alternatively called the A-*value* of t. Considering t as a mapping, the A-value of t is $t(A)$. Interpreting t as a row in a table, the A-value of t is the entry of t in the column headed by A. Since t is a mapping, we can restrict the domain of t. Let X be a subset of R. The usual notation for t restricted to X is $t_{|X}$. We, in our infinite knowledge, shall confuse the issue and write this restriction as $t(X)$ and call it the X-*value* of t. Technically, $t(A)$ and $t(\{A\})$ are different objects, but in keeping with the confusing customs of relational database theory, we often write A for the singleton set $\{A\}$. We also blur the distinction between $t(A)$ and $t(\{A\})$, even though one is just a value and the other is a mapping from A to this value. We assume there is some value λ such that $t(\emptyset) = \lambda$ for any tuple t. Thus $t_1(\emptyset) = t_2(\emptyset)$ for any tuples t_1 and t_2.

Example 1.2 Let t be the tuple defined in Example 1.1. The FROM-value of t is t(FROM) = O'Hare. The {FROM, TO}-value of t is the tuple t' defined by t'(FROM) = O'Hare, t'(TO) = JFK. We shall denote such a tuple as ⟨O'Hare:FROM JFK:TO⟩ or simply ⟨O'Hare JFK⟩ where the order of attributes is understood.

We have been treating relations as static objects. However, relations are supposed to abstract some portion of the real world, and this portion of the world may change with time. We consider that relations are time-varying, so that tuples may be added, deleted, or changed. In Table 1.1, flights may be added or dropped, or their times may be changed. We do assume, though,

*Actually, a tuple could be a multiset (a set with duplicates) of values, if domains for different attribute names intersect.

that the relation scheme is time-invariant. Henceforth, when dealing with a relation, we shall think of it as a sequence of relations in the sense already defined, or, in some cases, as potential sequences that the relation might follow, that is, possible states the relation may occupy. We shall discuss restrictions on the states a relation may assume, although nearly all of these restrictions will be *memoryless*: they will depend only on the current state of the relation and not on its history of previous states.

1.3 KEYS

A *key* of a relation r on relation scheme R is a subset $K = \{B_1, B_2, \ldots, B_m\}$ of R with the following property. For any two distinct tuples t_1 and t_2 in r, there is a $B \in K$ such that $t_1(B) \neq t_2(B)$. That is, no two tuples have the same value on all attributes in K. We could write this condition as $t_1(K) \neq t_2(K)$. Hence, it is sufficient to know the K-value of a tuple to identify the tuple uniquely.

Example 1.3 In Figure 1.1, {NUMBER} and {FROM, TO} are both keys.

Let us formulate some notation for relations, schemes, and keys. Our convention will be to use uppercase letters from the front of the alphabet for attribute symbols, uppercase letters from the back of the alphabet for relation schemes, and lowercase letters for relations. We denote a relation scheme $R = \{A_1, A_2, \ldots, A_n\}$ by $R[A_1A_2 \cdots A_n]$, or sometimes $A_1 A_2 \cdots A_n$ when we are not concerned with naming the scheme. (Another confusing custom of relational database theory is to use concatenation to stand for set union between sets of attributes.) A relation r on scheme R is written $r(R)$ or $r(A_1A_2 \cdots A_n)$. To denote the key of a relation, we underline the attribute names in the key. Relation r on scheme $ABCD$ with AC as a key is written $r(\underline{A}B\underline{C}D)$. We can also incorporate the key into the relation scheme: $R[\underline{A}B\underline{C}D]$. Any relation $r(R)$ is restricted to have AC as a key.

Example 1.4 We can write the relation scheme for the relation in Table 1.1 as FLIGHTS [NUMBER FROM TO DEPARTS ARRIVES].

If we wish to specify more than one key for a scheme or relation, we must list the keys separately, since the underline notation will not work. The keys explicitly listed with a relation scheme are called *designated keys*. There may be keys other than those listed; they are *implicit keys*. Sometimes we distinguish one of the designated keys as the *primary key*.

Our definition of key is actually a bit too broad. If relation $r(R)$ has key K', and $K' \subseteq K \subseteq R$, then K is also a key for R. For tuples t_1 and t_2 in r, if $t_1(K') \neq t_2(K')$, then surely $t_1(K) \neq t_2(K)$. We shall restrict our definition slightly.

Definition 1.1 A *key* of a relation $r(R)$ is a subset K of R such that for any distinct tuples t_1 and t_2 in r, $t_1(K) \neq t_2(K)$ and no proper subset K' of K shares this property. K is a *superkey* of r if K contains a key of r.

The new definition of superkey is the same as the former definition of key. We shall still use the old definition of key in designated key, that is, a designated key may be a superkey.

Example 1.5 In Table 1.1, {NUMBER} is a key (and a superkey), so {NUMBER, FROM} is a superkey but not a key.

There are some subtleties with keys. As we mentioned in the last section, we consider relations to be time-varying. For any given state of the relation, we can determine the keys and superkeys. Different states of the relation may have different keys. We consider relation schemes, though, to be time-invariant; we would like the keys specified with relation schemes not to vary either. Thus, in determining keys for a relation scheme, we look across all states a relation on the scheme may assume. Keys must remain keys for all permissible data.

Example 1.6 In Table 1.1, {FROM, TO} is a key for the relation. However, it is likely that there could be two flights between the same origin and destination, although they would undoubtedly leave at different times. Hence {FROM, TO, DEPARTS} is a key for the relation scheme FLIGHTS.

We shall mainly concern ourselves with keys and superkeys of relation schemes, thinking in terms of all permissible states of a relation on the scheme. What is and is not a key is ultimately a semantic question.

1.4 UPDATES TO RELATIONS

Now that we have relations, what can be done with them? As noted, the content of a relation varies with time, so we shall consider how to alter a relation. Suppose we wish to put more information into a relation. We perform an *add*

operation on the relation. For a relation $r(A_1A_2 \cdots A_n)$, the add operation takes the form

$$\text{ADD}(r; A_1 = d_1, A_2 = d_2, \ldots, A_n = d_n).$$

Example 1.7 Call the relation in Table 1.1 *sched*. We might perform the update

ADD(*sched*; NUMBER = 117, FROM = Atlanta, TO = Boston,
DEPARTS = 10:05p, ARRIVES = 12:43a).

When there is an order assumed on the attribute names, the shorter version

$$\text{ADD}(r; d_1, d_2, \ldots, d_n)$$

suffices.

Example 1.8 The short version of Example 1.7 is

ADD(*sched*; 117, Atlanta, Boston, 10:05p, 12:43a).

The intent of the add operation is clear, to add the tuple described to the relation specified. The result of the operation might not agree with the intent for one of the following reasons:

1. The tuple described does not conform to the scheme of the specified relation.
2. Some values of the tuple do not belong to the appropriate domains.
3. The tuple described agrees on a key with a tuple already in the relation.

In any of these cases, we consider $\text{ADD}(r; d_1, d_2, \ldots, d_n)$ to return r unchanged and in some manner indicate the error.

Example 1.9 If *sched* is the relation in Table 1.1, then

ADD(sched; NUMBER = 117, FROM = Atlanta, TO = Boston,
DATE = 4 March)

is disallowed for reason 1 above. The operation