Textbook Treatments of Relational Algebra

Hugh Darwen and C.J. Date

This is a companion paper to reference [1], to which it was originally planned as an appendix.

We have examined the various treatments of relational algebra in fifteen books in our own collections. We present our findings in a somewhat roughly annotated bibliography consisting of fifteen numbered entries (we use that term rather than "sections" because our text frequently refers to sections in the books we are commenting on).

Preliminary remarks:

Notes by HD/CJD: Each book was examined by just one of us, possessing a copy of the book in question, who provided notes on which the annotations are based. We indicate which of us provided the notes to avoid any confusion arising from the occasional use of the first person singular. We both occasionally express an opinion, sometimes favourable, sometimes not.

Restriction vs. selection: Some books use the term "selection" and its derivatives in connection with the operator called restriction. In these notes we use the term restriction except when referring to text that uses selection.

Relation comparisons: The treatment of operators such as "=" and "⊆" for comparing relations gets hardly any mention at all in the textbooks we have examined. They are useful in integrity constraint definitions and also in queries involving universal quantification, for example. As noted by Chris Date in reference [3], Codd's original papers make no mention of these operators. We have found no discussion of relation comparison operators in any of the fifteen books we have examined, apart from a very brief mention in Codd's book (entry number 15 in the list). They are covered in the sixth and subsequent editions of Chris Date's book (entry number 1, which deals with the third edition).

Common problems:

- Many books are unclear as to what exactly constitutes the relational model. In particular, there is a common tendency to treat the relational model and relational algebra as separate entities, the former dealing only with structure.
- The distinction between relations and relation variables is invariably somewhat blurred. Indeed, none of the books in this study uses the term relation variable (though it does appear in the sixth and subsequent editions of Chris Date's book). Several of the books advance the notion that a relation has a name (in addition to a heading and a body) and their account of the relational algebra pays some homage to that idea.
- Almost every account of the relational algebra omits both the extension operator and aggregation.
- Very few authors display awareness of the existence of ISBL and PRTV.

1. **C.J. Date:** ***An Introduction to Database Systems*** (3rd edition, Addison-Wesley, 1981; 5th edition, 1990; 6th edition, 1995; 8th edition, 2004). Notes, by HD, are on the 3rd edition, 574 pages, with occasional remarks on the other referenced editions.

This book eventually ran to an eighth edition and has been perhaps the best known and mostly widely used text on the subject since its first publication in 1975. It is almost certain that the authors of the other books in this study, apart from Codd himself, were directly or indirectly influenced by this book. From the fifth edition onwards the treatment of relational algebra is consistent with ISBL and eventually uses **Tutorial D**, a language jointly invented by Date and myself. What a pity it is, then, that none of the post-1990 authors cited here appears to have noticed that significantly improved treatment.

The story of how my collaboration with Date started (in 1988) is relevant here. In December 1987 I was able to engage both him and Ted Codd in conversation at a conference in London, England, run by their company. They had both given presentations criticizing SQL, but those criticisms did not mention SQL's failure to properly address the issue of unique column names and its resulting deficiencies. I described how we had addressed the issue in BS12, based on ISBL's treatment, and they both showed interest, inviting me to contribute an article to their monthly journal. In the event I, under the name Andrew Warden, contributed a short series of articles and my collaboration with Date started when he asked if I would be prepared for those articles to be included in one of his books, which appeared in 1990 [2]. Codd reviewed those articles but alas they had no effect on the book by him that is the subject of the last entry in this selection.

Back to Date's third edition: PRTV gets a mention in annotations to several references at the end of Chapter 12, **Relational Algebra**, but does not get a chapter in the series of seven on implementations of the relational model. Of those seven chapters, the first six are devoted to IBM's research prototype of the 1970s, System R. They include one on the language it spawned, SQL. The seventh is on Query By Example (QBE).

The relational operators described in Chapter 12 are union, intersection, difference, extended cartesian product, selection, projection, equijoin, greater-than join, natural join, and division (Codd's). There is no mention of extension, summarization, or relations of degree zero. A selection condition is "a boolean combination of terms, each term being a simple comparison", rather than being just an unrestricted boolean expression (think of something like IS_NUMERIC(X), where X is a character string). For union, intersection, and difference, "the two operand relations must be *union-compatible:* that is, they must be of the same degree, *n* say, and the *j*th attributes of the two relations (*j* in the range 1 [sic] to *n*) must be drawn from the same domain (they need not have the same name)". Earlier, in Chapter 4, **Relational Data Structure**, a relation is defined as a set of ordered *n*-tuples but the order in question is later described as "irrelevant", considering that "users normally refer to columns by name". In that chapter Date wrote: "In this book we shall generally assume that column ordering is insignificant unless we explicitly state otherwise." The text on union compatibility doesn't seem to explicitly state otherwise.

Aside from that infelicity concerning union compatibility, this book gets credit for being the only one in my collection to recognize the need for rules regarding attribute names and the blatant omission of such rules from Codd's work. However, instead of ISBL's renaming operator Date proposed a syntactic construct for defining an "alias" for a "declared relation", having much the same effect as a range variable introduced in Codd's calculus or an SQL FROM clause. He made a careful distinction between qualified and unqualified names, such that if R is "the name of a declared relation" with an attribute

named A, then R.A and A are both names for that attribute, but if R is joined with some other relation that also has an attribute named A, then the qualified name R.A must be used to refer to the one that originates from R, to avoid ambiguity.

For union, intersection, and difference, Date required the result to have "the same qualified attribute names as the first operand"—unlike SQL, which drops the qualifiers. The apparent consequence of the possible multiplicity of successive qualifiers arising in complex expressions isn't noted.

On the join operators, Date recognized the "important special case of natural join" and carefully defined it to be applicable only when the attributes whose *unqualified* names are common to both operands are based on the same "underlying domain". The operator name JOIN denotes natural join and this is the only join operator used in the illustrative examples. It's not clear from Date's definition how *r* JOIN (*s* UNION *t*) would be defined, considering that *s* might have attributes whose *qualified* names are required to appear in the result of *s* UNION *t*.

To illustrate the significant—one might even say dramatic—improvements introduced in the fifth edition, here are a couple of pertinent quotes from that edition:

> In the previous section we briefly discussed the importance of the relational closure property. However, there was one very significant point that we deliberately glossed over in that discussion. As explained in Chapter 11, a relation has two parts, a heading and a body; the heading is the set of attribute names and the body is the data, loosely speaking. Now, every *named* relation (i.e., every relation—base relation, view, etc.—that is explicitly defined in the database definition) will obviously have a proper heading; but what about unnamed (i.e., result) relations? For example, consider the equijoin of suppliers and parts over matching cities … What is the heading for that equijoin? Closure dictates that it must *have* a heading, and the system needs to know what it is. [An accompanying footnote describes the problems arising from SQL's treatment of that equijoin, giving two columns named CITY.]
>
> Our version of the relational algebra is … defined in such a way as to guarantee that *all* relations do have a proper heading—i.e., headings in which each attribute does have a proper unqualified name that is unique within its containing relation. [An accompanying footnote remarks that "this aspect has been overlooked in most treatments of the algebra—with the notable exception of the treatment found in Hall et al. [6] and Warden", adding an acknowledgement that his own treatment in previous editions had been very much inferior to "Warden's approach" (really ISBL's, I hasten to add!)]

And the algebra does now include RENAME, EXTEND, and SUMMARIZE (ISBL's $, for aggregation) as well as "generalized division" (Todd's version, actually not strictly a generalization of Codd's, as we later discovered). Relations of degree zero are noted in passing in the description of projection, where the reader is referred to an answer to one of the exercises.

Chapter 12 in the third edition was followed by Chapter 13, **Relational Calculus**. In the fifth edition this chapter, now Chapter 14, contains two tables depicting sample tuples of a putative cartesian product of relations S, P and SPJ with common attributes such as S#, P# and J#. In both tables there are two columns headed by the name S#, two headed by P#, two headed by J# and three headed by CITY. The text following the first table includes this:

> *Note:* … [W]e have not bothered to rename attributes (as we really ought to have done, to avoid ambiguity), but instead are relying on attribute position to show (e.g.) which "S#" comes from relation S and which from relation SPJ.

But Chapter 14, which includes a BNF grammar, does not show how attribute renaming might be achieved using the calculus notation. In the sixth edition the BNF grammar was silently revised to include a construct similar to SQL's "AS *column-name*".

2. **David Maier:** ***The Theory of Relational Databases*** (Computer Science Press, 1983). 637 pages. See http://web.cecs.pdx.edu/~maier/TheoryBook/TRD.html for a free download. Notes by HD.

This is quite good in my opinion. Considering when it was written, it's surprising (and upsetting) that so many other texts didn't follow its example. The relevant chapters are Chapter 2, "Relational Operators", Chapter 3, "More Operators on Relations", and Chapter 15, "Relational Query Languages". Chapter 15 starts with a fairly lengthy treatment of ISBL.

The only major problem I find with this text is the complete omission of extension and summarization, even from the section on ISBL in Chapter 15. My guess is that Maier had studied reference [6] and perhaps an early version of ISBL before those operators had been introduced. The evidence for a study of [6] is in Chapter 14, "Miscellaneous Topics" which includes a long, elaborate section on "computed relations"—the term used in [6]—using an example based on computing durations of air flights. It notes the apparent need for a join with an infinite relation (as suggested in [6] and in our algebra **A** [4]) but suggests no method of supporting that need by introducing a new relational operator (viz. EXTEND).

In Chapter 2, **2.1 BOOLEAN OPERATIONS** deals with union, difference, and intersection, each of which is defined for pairs of relations "on the same scheme", where a scheme has been defined in Chapter 1 as a set of attribute names. Each attribute name is said to have a "corresponding" domain but the text is unclear on whether "same scheme" is meant to imply that attributes of the same name must also have the same domain. Chapter 1 is very clear that there is no ordering to the attribute values of a tuple, but the point is not mentioned again in this section, where it becomes especially important.

Maier mentions "antijoin" in Chapter 15, the section on ISBL, referring to ISBL's generalization of difference (NOT MATCHING in **Tutorial D**). The mention suggests to me that this term, which came my way only quite recently, is indeed to be attributed to Maier.

Curiously, the bulk of Section 2.1 is devoted to a discussion of relational *complement*. Correctly rejecting the operator (for the reasons we know well), Maier introduces in its place an operator he calls the *active complement* of relation *r*, being the join (cartesian product) of the projections of *r* on each attribute of *r* in turn (though not defined that way because he hasn't dealt with project and join yet).

2.2 THE SELECT OPERATOR restricts the selection condition to expressions of the form *attr-name* = *literal*. Most of the section is devoted to the associative and distributive

properties of this operator, ending with a curious note to the effect that it does not distribute over "complementation" (meaning his *active complement* operator).

2.3 THE PROJECT OPERATOR descends rather mysteriously into sloppiness ("the rows in a relation", "… obtained by striking out columns corresponding to attributes …"), but is essentially correct. Unsurprisingly, no mention is made of relations of degree zero, nor of an alternative notation for projection, by attribute exclusion.

2.4 THE JOIN OPERATOR is natural join, denoted $r \bowtie s$.

Chapter 3 starts with a lengthy section on DIVIDE (Codd's version), which is followed by a section titled CONSTANT RELATIONS, giving a notation for relation literals. Next comes the all-important **3.3 RENAMING ATTRIBUTES**, introducing the operator for that purpose and giving examples of its practical use. The remaining sections devote rather too much text (in my opinion) to equijoin, theta-join, theta-select, and a couple of operators, SPLIT and FACTOR, that operate on a single relation and yield a pair of relations. The term relational completeness isn't used but Maier offers the following list as a "restricted set that has all the power of the full set": single-attribute, single-tuple constant relations, select with a single comparison, natural join, project, union, difference, and renaming.

3. **Jeffrey D. Ullman: *Principles of Database Systems*** (2nd edition, Computer Science Press, 1982). 484 pages. Notes by HD.

There's a section on ISBL but it completely omits mention of extension (#) and renaming (%%) and incorrectly claims: "The ISBL language is fairly limited, when compared with query languages to be discussed in the next sections. For example, it has no aggregate operators (e.g., average, min)…". The book's references include [6], but that paper doesn't mention # and $, instead describing the "join-with-infinite-relation" approach to extension.

Ullman's operators are *Union, Set difference, Cartesian product, Projection, Selection* to begin with, followed by a section headed **Some Additional Algebraic Operations** that mentions *Intersection, Quotient, Join, Natural Join*. As usual, extension and summarization are missing (consistent with his outrageous treatment of ISBL), as are renaming, antijoin, semijoin, projection by exclusion, and relations of degree zero.

The brief description of *Union* has: "We only apply the union operator to relations of the same arity [the degree], so all the tuples of the result will have the same number of components."

For *Natural Join* we see that it "is applicable only when both [operands] have columns that are named by attributes". The definition is accurate as far as it goes (no mention of the requirement for common attributes to be of the same domain) but uses dot qualification, saying "Recall that *R.A* is the name of the column of $R \times S$ corresponding to the column *A* of *r* and *S.A* is defined analogously." $R \times S$ is cartesian product, so is apparently defined even when the operands have common attributes.

4. **Peter Gray:** ***Logic, Algebra and Databases*** (Ellis Horwood, 1984). 294 pages. Notes by CJD.

PRTV is mentioned and appears to have been somewhat influential.

This book is something of ray of light, which can perhaps be accounted for by the fact that Gray was (and is) based at Aberdeen University in Scotland and was in fairly close contact with Stephen Todd—Gray visited Peterlee and Todd visited Aberdeen.

Chapters 6-9 are "The Relational Model," "Calculus-Based Languages," "Relational Algebra: An Applicative Language," and "Query Transformation in Algebra and Calculus," respectively (4 chapters out of a total of 12).

Chapter 6 unfortunately suggests that tuples of a relation have a top to bottom order (page 116) and left to right attribute order (page 119, though this appears to be contradicted on page 127). Domains are basically equated to types. There is no distinction between relations and relation variables. A primary key is required for every "relation". Section 6.6 gives a preliminary discussion of relational operators, mainly for the purpose of explaining the decomposition and recomposition processes in normalization. This includes decomposing by selection and recomposing by union, as well as the usual projection and join. The discussion on join mentions join as an intersection, join as a generalised intersection (the term used in HHT1975), join as combined intersection and product, join as cartesian product, equijoin, and natural join "which we shall just call join". Outer join is also mentioned, as "suggested by Codd" (actually it was Ian Heath).

Chapter 8 presents a relational algebra based on the language ASTRID by Gray and D.M.R. Bell, stating this to be "strongly influenced by PRTV". The introductory text mentions "extra operations, principally Extend and Group-By, which were not in Codd's algebra".

The detailed text describes the operators mentioned below, though the Astrid grammar given in Appendix II uses capital letters. Square brackets are part of the syntax, not for optionality.

rx **selected_on [*bx*]** is restriction, where *bx* is a boolean expression.

rx **extended_by [*acl*],** where *acl* is an <*assign commalist*> as in **Tutorial D**, is extension.

rx **projected_to *A1*, *A2*, ..., *An*** is projection, and ***rx*** **discarding *A1*, *A2*, ..., *An*** is a projection by exclusion, this being the only book we have found to mention that alternative. But in both cases *n* is required to be greater than zero and there's no mention of relations of degree zero.

rx1 **union *rx2*** is union, which is "only possible if the two tables are of the same type, i.e. they have the same column names and column types. We then say they are 'union compatible.'"

rx1 **joined_to *rx2*** is (natural) join.

rx1 **without *rx2*** is difference.

rx1 **produced_with *rx2*** is cartesian product.

rx1* intersect_with *rx2 is intersection.

rx1* divided_by *rx2 appears to be Codd's divide but is not properly discussed.

***rx* group_by [*A1*, *A2*, ..., *An* creating *B1* := *x1*, *B2* := *x2*, ..., *Bm* := *xm*]** is SUMMARIZE … BY, supporting aggregation specifications COUNT, SUM, MAX, MIN, ANY, ALL but apparently no AVG.

rx* renaming *A1* as *B1*, *A2* as *B2*, …, *An* as *Bn is attribute renaming.

5. **Ramez Elmasri and Shamkant B. Navathe: *Fundamentals of Database Systems*** (6th edition, Addison-Wesley, 2010). 637 pages. Available as a free download from http://www.cvauni.edu.vn/imgupload_dinhkem/file/CSDL/Fundamentals_of_Database_Systems,_6th_Edition.pdf. The first edition is dated 1989. Notes by HD.

Extension and summarization are omitted.

SELECT is theta-select extended by use of logical connectives.

PROJECT is correctly defined but the text refers to the operand as a table and uses both terms column and attribute! No mention of projection by exclusion or relations of degree zero.

RENAME can "rename" the relation as well as or instead of its attributes. The relation name can be used as a "dot qualifier" for attribute names, in SQL-like fashion.

UNION, INTERSECTION, and MINUS require operands to "have the same type of tuples". They don't mention antijoin.

CARTESIAN PRODUCT (×), JOIN, theta-JOIN, EQUJOIN, NATURAL JOIN all given individual treatment. JOIN is denoted $r \bowtie_{cond} s$, where *cond* is a condition as in SELECT (so their ⋈ isn't natural join!). They don't mention that if natural join is available, then all the others, plus intersection, are redundant.

DIVISION is Codd's.

{SELECT, PROJECT, UNION, MINUS, RENAME, CARTESIAN PRODUCT} is offered as a "complete set".

6. **Paolo Atzeni and Valeria De Antonellis: *Relational Database Theory*** (The Benjamin/Cummings Publishing Company, 1993). 389 pages. Notes by HD.

A *relation scheme* is defined as "a name (*relation name*) together with a set of distinct attribute names".

Relational algebra is described as a "procedural language", one whose "expressions describe, step by step, the computation of the result from a database instance".

The treatment is good, as far as it goes, but extension and summarization are not mentioned.

They define union, intersection, difference, projection, selection, renaming, and join. The "set" operators require operands to be "defined on the same set of attributes", noting that "[o]ther authors propose a looser restriction, requiring the operands to have only the same degree; then the attributes of the result are either undefined or coincide with those of the

first operand. The present approach guarantees uniformity and symmetry, and it does not limit the expressive power, due to the introduction of the renaming operator". They do not give a justification for that "introduction", but we can assume they mean that these "other authors" do not include renaming in their algebras.

Selection is theta-select extended by use of logical connectives.

The definitions are consistent with ISBL; join is natural join. They don't mention antijoin. They do mention cartesian product as being a special case of join, but not that intersection is too. They mention theta-join as being "usually included in relational algebra" but they point out that it is redundant.

On renaming, they write: "The renaming operator overcomes the rigidity of those operators (such as natural join and the set-theoretic ones) that rely heavily upon attribute names."

7. **Catherine M. Ricardo:** ***Database Systems: Principles, Design, and Implementation*** (Macmillan, 1990). 576 pages. Notes by HD.

Neither ISBL nor PRTV appears in the index.

The section on relational algebra starts: "The relational algebra is a theoretical language with operators that work on one or more relations to define another relation without changing the original relation(s)."

For "the SELECT operator" Ricardo gives, as "the general form:

```
SELECT table-name WHERE condition [GIVING new-table-name]
```

Then, "[s]ymbolically, the form is $\sigma_{predicate}$(table-name)". Oh dear—so the expression denoting the relation operand has to be just a name and cannot be a more general expression. A similar comment applies to the other relational operators

In all, Ricardo defines SELECT, PROJECT, TIMES, THETA JOIN, EQUIJOIN, JOIN, SEMIJOIN, OUTERJOIN (several varieties), UNION, DIFFERENCE, INTERSECTION, DIVIDEBY, all spelt thus. So, again, no extension, no summarization, no antijoin to go with semijoin, and *no renaming*.

JOIN is natural join, with the following curious comment: "Although normally the JOIN is performed over columns with the same names, all that is really required is that the domains of the attributes be the same."

As usual, there is no mention of projection by exclusion or relations of degree zero. Operands of union, intersection, and difference "must be **union compatible**. This means that they must have the same basic structure. In particular, they must have the same degree and the attributes *in the corresponding position* in both relations must have the same domains." [My italics. Oh dear again.]

8. **Gottfried Vossen:** ***Data Models, Database Languages, and Database Management Systems*** (Addison-Wesley, 1990). 590 pages. Notes by HD.

No mention of ISBL or PRTV.

Vossen gives projection, selection, intersection, union, difference, and *n*-adic natural join, claiming this to be a "complete set". Later he mentions "other relational operations (like

division, total projection, equi-, theta-, or semi-join, renaming" as dispensable, "since a 'complete' set of operations … is already at our disposal." He notes that join is commutative and associative, adding: "it degenerates to a Cartesian product (intersection) if the sets of attributes of the operands are disjoint [(equal)], respectively". (That "(equal)" is missing in the book. Perhaps it was intended but got lost in the production process. The sentence makes no good sense without it.)

It's not clear what operator he means by "renaming". Attribute renaming is dispensable if extension and projection are supported but can't be defined in terms of the set of operators he claims to be complete.

He makes clear that the attributes of a relation are not ordered and requires the operands of union to be "of the same format", without further comment.

As usual, there is no extension, no summarization, no mention of relations of degree zero.

9. **Patrick O'Neil:** ***Databases Principles Programming Performance*** (Morgan Kaufmann, 1994). 874 pages. Notes by CJD.

The book title is apparently unpunctuated as shown. No mention of ISBL or PRTV.

Chapter 2 is "The Relational Model". Sections 2.5, 2.6, 2.7, 2.8, and 2.10 are "Relational Algebra," "Set-Theoretic Operations," "Native Relational Operations," "The Interdependence of Operations," and "Other Relational Operations," respectively. Relational calculus is not mentioned.

Relations have headings—the text does use that term—but the components of a heading are referred to as "columns". We can perhaps assume that column *names* are meant but text is still unclear as to whether a heading consists of just column names or of column-name:domain-name pairs. In any case little attention is given to the heading concept in the material on relational operators, which is presented using both symbols and keywords.

The important property of closure is treated in a rather casual, not to say sloppy, manner:

> Information is stored [*sic*] in a relational system in the form of tables, so it seems natural [*sic*] to express the results [*sic*] of a query in table form.

And later:

> ... relational operations can be applied recursively [*sic*], so that [input] tables ... can themselves be the results of other relational algebra expressions.

However, the author does mention one important property of relations that many others overlook:

> It is worth mentioning here that the relational model also states that there should be no order to the *columns* of a relation, so once again our mental model of a table is slightly misleading. This rule is broken, however, by the standard SQL language, as we will see in Chapter 3.

Union, intersection, and difference are described using set theory symbols or the keywords UNION, INTERSECT, MINUS. The requirement for operands to be of the same type is called compatibility. Relations are compatible if they have the same

heading, this being also the heading of the result, but, as we have seen, it is not clear that columns of the same name must also be of the same domain.

Projection is described using square brackets. The result heading is defined in the obvious way.

Attribute renaming is supported only by an operator that looks like assignment. It uses the assignment symbol, :=, and the description says it "allows a redefinition of all attribute names in the heading of the original table". (Notice that columns have now become attributes.) For example, S[B1,...,Bn] := R[A1,..,An]. This appears to assume that the source relation has a name and cannot be denoted by some arbitrary expression. Notice also that one apparently has to provide names for all the attributes, though presumably they don't all have to be different from their input counterparts. In any case, the attribute renaming part is given as optional—we can write just S := R, when S is then described as an **alias** of R. There seems to be some confusion here. Also it is not clear as to whether such assignments are separate statements or just part of an expression.

Restriction is called selection and uses the syntax *R* WHERE *C*. The condition *C* is limited to being constructed from terms of the form "attribute *op* attribute" or "attribute *op* constant" and the logical connectives AND, OR, and NOT. The result heading is clearly the same as that of the input, though the text doesn't actually say as much

Cartesian product is described using cross × or TIMES. No compatibility requirement is mentioned and the result apparently has dot-qualified column names. Moreover, the operators described so far, including :=, are claimed to be a "basic set", suggesting that they collectively provide some form of completeness:

> We claim that a basic set of operations consists of union, difference, product, selection, and projection, together with the assignment operator, which allows us to redefine attribute names.

Join is described using the bow tie ⋈ or JOIN and we are told to note "that the join operation defined here is also known as *equijoin* or *natural join*", betraying a gap in the author's understanding of those terms. However, TIMES and INTERSECT are correctly noted as special cases. Left, right, and full outer join and theta join are also described, but not semijoin.

Relational division, using the syntax $A \div B$, is basically Codd's original operator of that name.

As usual, there is no extension, no summarization, and no mention of relations of degree zero.

10. **Serge Abiteboul, Richard Hull, and Victor Vianu** *Foundations of Databases: The Logical Level* (Addison-Wesley, 1994). A free PDF copy, excluding some of the front matter, is obtainable at http://webdam.inria.fr/Alice/. 678 pages in the PDF but highest page number is 685. Notes by HD.

This one is highly theoretical in its approach and is definitely not for the faint of heart. The front cover of the printed book shows a picture of Lewis Carroll's Alice and each chapter carries an epigraph in the form of an imaginary conversation between Alice and the authors.

Neither ISBL nor PRTV appears in the index.

In spite of my sprinkling of snide comments in these notes, the coverage of relations and relational algebra is a worthy effort, justifying my rather more detailed treatment compared with most of the other books discussed in this paper.

In Chapter **3, The Relational Model,** the authors take the trouble to explain what they mean by that term:

> The term relational model is actually rather vague. As introduced in Codd's seminal article, this term refers to a specific data model with relations as data structures, an algebra for specifying queries, and no mechanisms for expressing updates or constraints. Subsequent articles by Codd introduced … the first integrity constraints for the relational model—functional dependencies. … Soon thereafter, researchers in database systems implemented languages based on the algebra and calculus, extended to include update operators and to include practically motivated features such as arithmetic operators, aggregate operators, and sorting capabilities. … The term relational model has thus come to refer to the broad class of database models that have relations as the data structure and that incorporate some or all of the query capabilities, update capabilities, and integrity constraints mentioned earlier. In this book we are concerned primarily with the relational model in this broad sense.

The terminology introduced in Section **3.1. The Structure of the Relational Model** is somewhat idiosyncratic. Having first given the intuitive terms table, row, and column they then give informal definitions that appeal to those terms, getting off to an infelicitous start with: "Each table is called a relation and it has a name". It is not clear what the name is for and not all relations in their later examples appear to have names. Then:

> The columns also have names, called attributes (e.g, Title). Each line in a table is a tuple (or record). The entries of tuples are taken from sets of constants, called domains, that include, for example, the sets of integers, strings, and Boolean values.

So now a table has lines instead of the previously mentioned rows. And that needless "(or record)" can surely only create further confusion. The text continues:

> Finally we distinguish between the database schema, which specifies the structure of the database; and the database instance, which specifies its actual content. This is analogous to the standard distinction between type and value found in programming languages (e.g., an identifier X might have type *record A : int, B : bool endrecord* and value *record A : 5, B : true endrecord*).

That last sentence would seem to lie better if it immediately followed the sentence starting "The entries of tuples". When it comes to the formal definitions, we see the term *arity* in place of the more usual term *degree* that Codd introduced; and we see *sort* in place of *heading*, though the attributes constituting a sort are just names, as previously indicated, rather than <name, domain> pairs. True to the concept of every relation having a name, a sort is said to apply to a relation name, though it's not clear that the authors stick to this notion in their text on relational algebra.

The authors give a notation for tuple literals, explicitly including the 0-tuple, but they make no mention of relations of arity (i.e., degree) zero. They do not immediately proceed to give notation for relation literals but such notation is silently introduced in at least one example much later.

In Section **3.2, Named and Unnamed Perspectives,** the authors make a clear and laudable distinction between relations consisting of ordered *n*-tuples (the "unnamed perspective")

whose elements are referred to by their ordinal position, and relations consisting of tuples that are not ordered and whose elements are referred to by names (the "named perspective").

The named/unnamed "perspective" distinction leads to two differing accounts of relational algebra in Chapters **4, Conjunctive Queries,** and **5, Adding Negation: Algebra and Calculus.** (It's curious that *dis*junction—via the union operator—is covered in Chapter 4 in spite of the title but negation—via the difference operator—is deferred to a separate chapter.)

After a lengthy preliminary discussion with lots of illustrations, Section **4.4, Algebraic Perspectives,** offers two distinct but equivalent algebras. The first, called SPC, is introduced, in a subsection headed **The Unnamed Perspective: The SPC Algebra,** by

> Three primitive algebra operators form the *unnamed conjunctive algebra*: selection, projection, and cross-product (or Cartesian product). This algebra is more often referred to as the *SPC algebra*, based on the first letters of the three operators that form it.

The reader might be more familiar with those italicized terms than I am, but they are perhaps useful. It is of course quite normal to describe a relational algebra in an order determined by conjunction, disjunction, negation, but why is projection (i.e., existential quantification) included in SPC, considering that it has nothing to do with conjunction?

The notation uses the familiar Greek letters and subscripts. Here is Example 4.4.1, illustrating select:

$$I_1 := \sigma_{2=\text{“Bergman”}}(Movies)$$

The number 2 refers to the second "column", that term presumably being used because "attribute" has been defined as a name. For a similar presumed reason, the term *arity* is used for headings, the previously defined *sort* (for heading) being a set of attributes (i.e., names). Notice the inclusion of a name, I_1, consistent with the earlier definition of relation. This consistency is not maintained throughout the chapter (see my next example), so we have to wonder what the assignment really means. The selection condition is initially restricted to a simple comparison but later extended to allow comparisons to be connected by the usual logical operators.

Projection uses the symbol π with a subscripted list of column numbers. The following example combines projection with selection and cross-product ($\times$).

$$I_5 := \pi_{2,3}(\sigma_{1=2}(I_4 \times Location))$$

The name I_5 is given for the final result, I_4 being the name assigned to a previous example, but the relations denoted by the invocations of $\times$ and σ have no names, thus contravening the definition given for relation in Chapter 3.

Intersection and equijoin are mentioned as common additions to SPC that "can be simulated by the primitive ones" but semijoin and composition are not.

The next section is headed **The Named Perspective: The SPJR Algebra**. The letters stand for select, project, join (natural join, using $\bowtie$), and rename. Headings are now sorts and attributes replace the use of column numbers. Example 4.4.3 includes the following expression where even the final result is a relation with no name:

$$\pi_{Theater,Address}((\sigma_{Director=\text{“Bergman”}}(Movies) \bowtie Pariscope) \bowtie Location)$$

Attribute renaming uses the symbol δ. For example, $\delta_{BC \to DA}(r)$ denotes a renaming of relation *r* such that attribute B is renamed D in the result and C is renamed A.

No mention is made of the additional operators intersection and equijoin that were proposed for the SPC algebra; semijoin and composition remain absent.

The relational union operator is introduced in Section **4.5, Adding Union**. Both algebras are augmented by this operator, yielding new algebras called SPCU and SPJRU. In SPCU the operands of union must be of the same arity; in SPJRU they must be of the same sort. A slightly unfortunate example has been chosen to illustrate this operator:

$$\pi_{Theater}(\sigma_{Title=\text{“Annie Hall”}}(Pariscope) \cup \sigma_{Title=\text{“Manhattan”}}(Pariscope))$$

giving theaters where either of the movies Annie Hall or Manhattan can be seen. They point out that this can also be done using a single invocation of σ:

$$\pi_{Theater}(\sigma_{Title=\text{“Annie Hall”} \vee Title=\text{“Manhattan”}}(Pariscope))$$

and also by using ⋈ with a relation literal, denoted simply by braces surrounding a couple of tuple literals:

$$\pi_{Theater}(Pariscope \bowtie \{<Title\text{: “Annie Hall”}>, <Title\text{: “Manhattan”}>\})$$

So they don't actually give an example to make the case for requiring union in a relationally complete algebra.

Negation is dealt with in Chapter **5, Adding Negation: Algebra and Calculus.** The algebra bit is covered in three short paragraphs describing and illustrating the addition of the difference operator (−) to both SPCU and SPJRU. Antijoin is not mentioned. Curiously, this chapter includes Section **5.5, Aggregate Functions**. It's good to see some coverage of aggregation, but why in a chapter putatively devoted to negation? "Aggregate functions are added to the algebra using an extended projection operator", says the text. The general form is shown as

$$\pi_{j1,\ldots,jm;f(k)}(I)$$

where *I* is a relation, *f* is an aggregate function such as SUM and *k* is an attribute of *I*. The definition makes this look very much like the **Tutorial D** SUMMARIZE … BY operator (and ISBL's $), the attributes *j1, ..., jm* being the "BY" attributes. There is no discussion of basis operators, such as + for SUM, and therefore no mention of identity values and the treatment of aggregation over the empty set. Even more curiously, relational extension is not discussed at all, even though it could presumably have been done in similar fashion to aggregation by allowing expressions such as $j1 + j2 \to k$ in an invocation of π.

11. **Thomas Connolly and Caroline Begg: *Database Systems*** (Addison-Wesley, 3rd edition, 2002). 1236 pages. Notes by HD.

The first edition was published in 1995.

The definitions of relation, attribute, and tuple are informal and imprecise. The section headed **Properties of Relations** starts, unfortunately, with "the relation has a name that is distinct from all other relation names in the relational schema", but gets (a bit) better:

- each cell of the relation contains exactly one atomic (single) value
- each attribute has a distinct name
- the values of an attribute are all from the same domain
- each tuple is distinct; there are no duplicate tuples
- the order of attributes has no significance
- the order of tuples has no significance, theoretically [suggesting that it might have some significance "in practice"?]

Neither ISBL nor PRTV appears in the index.

The section on relational algebra starts: "The relational algebra is a theoretical language with operations that work on one or more relations to define another relation without changing the original relation(s)." (Now, where have I seen that before? Have a look at the second paragraph in entry number 7 on Catherine Ricardo's book: the cited text, published five years after Ricardo's, differs only in the word "operations", replacing "operators".)

They describe selection, projection, union, difference, intersection, cartesian product (which "multiplies two relations"—without the quotes!), theta join, equijoin, natural join, outer join, semijoin, division (Codd's), but no extension or summarization—and *no renaming*. There is no projection by exclusion and no mention of degree-zero relations.

For union and difference they require only that the operands be of the same degree and that "corresponding attributes" have the same domain, adding: "Note that attribute names are not used in defining union-compatibility". Words fail me.

12. Abraham Silberschatz, Henry F. Korth, and S. Sudarshan: *Database System Concepts* (4th edition, McGraw-Hill, 2002). 1064 pages. Notes by CJD.

Neither ISBL nor PRTV appears in the index but the bibliography includes Patrick Hall's "Optimisation of a Single Relational Expression" and Stephen Todd's "The Peterlee Relational Test Vehicle—A System Overview". It is not clear whether either of these papers is mentioned anywhere in the text.

Chapter 3, **Relational Model,** has Sections 3.1, **Structure of Relational Databases**, 3.2, **The Relational Algebra,** and 3.3, **Extended Relational-Algebra Operations**.

Section 3.1 contains this odd remark: "[A] tuple variable is a variable whose domain is the set of all tuples", and also:

> The concept of a relation corresponds to the programming-language notion of a variable. The concept of a **relation schema** corresponds to the programming-language notion of type definition ... The concept of a **relation instance** corresponds to the programming language [*sic no hyphen*] notion of a value of a variable ... However, we often say 'relation' when we actually mean 'relation instance.'

The relational algebra is described as "a *procedural* query language" as opposed to relational calculus, described as "a **nonprocedural** query language".

Select is the restriction operator, using σ, with a subscript for the boolean expression—which is limited to being constructed from terms of the form "attribute *op* attribute" or "attribute *op* constant" and the usual three connectives, where *op* is one of the usual six.

The "schema" (i.e., heading) of the result is not discussed. The examples illustrate—it has to be said—how very unwieldy this subscript or suffix style is.

Project uses Π, with a subscript for attribute names. Again the "schema" of the result is not discussed.

Union and **set-difference** use set theory symbols and require the operands to be compatible. Corresponding attributes in a left to right ordering must be defined on the same domains. Again the "schema" of the result is not discussed

Cartesian-product uses ×. A left to right ordering of attributes is assumed. The "schema" of the result has dot qualified names but there is a forward reference to renaming.

Rename uses ρ and follows Elmasri and Navathe (see entry number 5) in allowing relations to be renamed as well as attributes. Here's a revealing quote:

> The rename operation is not strictly required, since it is possible to use a positional notation for attributes ... The positional notation also applies to results of relational-algebra expressions.

Having affirmed that the above operators "are sufficient to express any relational-algebra query" the authors then define **set-intersection** and **natural-join** ⋈ and **theta join** and **division** ÷, to which they add **assignment** (using a left arrow), whose semantics are described as "work[ing] like assignment in a programming language".

There then follow some so-called "extended" operations. These do include support for extension and aggregation.

Generalized projection combines projection and extension, using introduced attribute names. Although PRTV had a similar operator, there's no mention of PRTV or ISBL in the index. Considering that introduced names for added attributes are optional, giving rise to unnamed attributes, it seems likely that SQL's SELECT clause was the inspiration for generalized projection.

Aggregate functions appear to be counterparts of SQL's constructs of the same name (in the ISO standard), such as SUM(X), COUNT(*) and so on, including DISTINCT variants, but using subscript syntax.

Aggregation is effectively the "BY" variant of **Tutorial D**'s SUMMARIZE, using introduced attribute names (optional again).

Outer join comes in several varieties using the bow tie ⋈.

Relational completeness is hinted at without much discussion.

There is no projection by exclusion, no semijoin or antijoin, no mention of degree-zero relations.

13. Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer Widom: *Database Systems: The Complete Book* (3rd edition, Prentice Hall, 2003). 1119 pages. Notes by CJD.

The treatment of the relational model is among the worst I have encountered. There is no mention of ISBL or PRTV. An index entry for Patrick Hall refers to a mention of his

paper “Optimisation of a Single Relational Expression” in the references for Chapter 16, “The Query Compiler”.

Chapter 3 is “The Relational Data Model”, sandwiched between Chapter 2, “The Entity-Relationship Model” and Chapter 4 “Other Data Models”, but it betrays a poor understanding of what the relational model is and is not. Section 3.1 “Basics of the Relational Model” is just four pages long, with no mention of either relational algebra or relational calculus, whereas sixty-two pages in six more sections are devoted to relational database design. This material on database design includes details of dependency theory and normal forms (but stopping at 4NF!) even though the decomposition and recomposition operators, join and project, are not properly explained until Chapter 5, “Relational Algebra”.

Relational algebra is dealt with in the 48 pages of Chapter 5, “Relational Algebra” (see next paragraph). Relational calculus is discussed in one section of Chapter 10, “Logical Query Languages”.

Chapter 5 discusses relational algebra, using the mathematical symbols $\cup$, $\cap$, -, $\times$, σ, π, ρ, and $\bowtie$ for union, intersection, difference, product, selection (i.e., restriction), projection, renaming, and join, respectively (with subscripts for restriction conditions and attribute name lists). The chapter is divided into seven sections, the operator descriptions appearing in Section 5.2, “An Algebra of Relational Operations”. The following extract is telling:

> Initially, relational algebra was proposed by T. Codd [*sic*] as an algebra on sets of tuples (i.e., relations) ... When DBMS’s that used the relational model were first developed, ... for efficiency purposes, [they] regarded relations as bags, not sets.

Union, intersection, difference: “*R* and *S* must have schemas with identical sets of attributes, and the types (domains) for each attribute must be the same in *R* and *S* ... [The] columns of *R* and *S* must be ordered so that the order of attributes is the same for both relations”. The result schema is not discussed but examples show it to be the same as that of both inputs.

Projection: As usual, only the “inclusion” form is described and the result must have at least one attribute (the book makes no mention of relations of degree zero). This time the result schema *is* defined (in the obvious way).

Selection: This term is used instead of restriction in spite of the fact that the authors appear to think that SQL DBMSs, where the keyword SELECT is used for a different purpose, were the first to use the relational model. Again, the result schema *is* defined (in the obvious way).

Cartesian product or **cross-product** or just **product:** The result has dot qualified attribute names, with attributes derived from the first operand first (in left to right order) followed by those of the second operand.

Natural join: The description fails to observe that product and intersection are special cases. **Theta join** is also described, using the bow tie with a suffix for the boolean expression, but neither semijoin nor antijoin are mentioned.

Renaming: "Does not affect the tuples of a relation but changes the relation schema, i.e., the names of the attributes and/or the name of the relation itself". Ouch!

Section 5.3 is sloppily titled "Relational Operations on Bags". A bag is apparently thought to be a special kind of set: "[If] a 'set' is allowed to have multiple occurrences of a member, then that set is called a *bag*". Or perhaps not, considering the quotes around the first appearance of the word "set" in that sentence.

Section 5.4 is "Extended Operators of Relational Algebra" but is mainly devoted to operations that are not operators on relations at all, such as duplicate elimination (the internal operation used by DBMSs to avoid repeated appearances of the same tuple), sorting (another internal operation that is commonly used in the evaluation of relational expressions), aggregation, and grouping. A form of extension is introduced under the name "extended projection", clearly inspired by SQL's SELECT clause. One would expect at least Codd's original "divide" operator to appear in this chapter, but it doesn't. There are numerous examples of the form "If *R* is the [following] relation ..." where *R* in fact contains duplicate tuples, showing how muddled the text is over what a relation is.

14. Raghu Ramakrishnan and Johannes Goehrke: *Database Management Systems* (3rd edition, McGraw-Hill, 2003). 1065 pages. Notes by CJD.

PRTV gets a brief mention in the Bibliographic Notes on Chapter 14, "Evaluating Relational Queries":

> The implementation techniques used in PRTV, which utilized relational algebra transformations and a form of multiple-query optimization, are discussed in [5].

and also in the Bibliographic Notes on Chapter 3, "The Relational Model", with reference to Stephen Todd's "The Peterlee Relational Test Vehicle" [7]:

> Pioneering efforts include [System R, Ingres, QBE, and] PRTV at the IBM Scientific Center [*sic*] in Peterlee

Chapter 3 is "The Relational Model". Chapter 4 is "Relational Algebra and Calculus", with Sections 4.1 "Preliminaries", 4.2 "Relational Algebra", 4.3 "Relational Calculus", and 4.4 "Expressive Power of Algebra and Calculus".

The algebra is described as procedural, the calculus declarative.

The text makes no clear distinction between relations and relation variables and is in fact quite confused in places:

> A query is evaluated using *instances* of each input relation and it produces an instance of the output relation.

This is like saying that an addition in arithmetic uses *instances* of each input number to produce an instance of the output number. What exactly is meant by "instance" becomes unclear when we come to

> A relation consists of a **relation schema** and a **relation instance.**

followed by

> The relation instance is a table ...

and later by

> The term *relation instance* is often abbreviated to just *relation* ... .

In Chapter 4 we find

> In defining relational algebra and calculus, the alternative of referring to fields by position is more convenient than referring to fields by name ... if we use field names to refer to fields, the definition of query language constructs must specify the names of fields for all intermediate relation instances. This can be tedious and is really a secondary issue, because we can refer to fields by position anyway.

which seems to rule out the commutativity of join. However, later we find

> We ... introduce simple conventions that allow intermediate relations to 'inherit' field names, for convenience.

Selection: Uses σ with a suffix for the boolean expression, which is restricted to being formed from terms of the form "attribute *op* attribute" or "attribute *op* constant", where *op* is one of =, <, >, >=, <=, <>. Terms can be combined using the connectives ∧ and ∨, but negation is not mentioned. Note that "fields" have suddenly become "attributes! "The schema of the result ... is the schema of the input relation instance."

Projection: Uses π with a suffix for attribute names. "The schema of the result ... is determined by the fields [*sic*] that are projected [*sic*] in the obvious way."

Union, intersection, difference: These use set theory symbols, ∪, ∩, and -. They require "union compatibility", where attributes correspond by left to right position and must be defined on same domains. "The schema of the result is ... identical to the schema of [the first operand]."

Cross-product: This depends on a left to right ordering and so is noncommutative, like its counterpart in mathematics. The description includes

> It is possible for both *R* and *S* to contain one or more fields having the same name; this situation creates a *naming conflict*. The corresponding fields in [the result] are unnamed and are referred to solely by position.

Notwithstanding that astounding observation, we later encounter in **4.2.3 Renaming**:

> It is ... convenient to be able to give names explicitly to the fields of a relation instance that is defined by a relational algebra expression. In fact, it is often convenient to give the instance itself a name so that we can break a large algebra expression into smaller pieces by giving names to the results of subexpressions.

And they go on to define an operator ρ that can perform both of these operations, as we have seen elsewhere.

That the authors regard each of the foregoing operators as essential for completeness is evidenced by

> It is customary to include some additional operators in the algebra, but all of them can be defined in terms of the operators we have defined thus far.

and those additional operators are:

Condition join: This is theta join. It uses the bow tie with a suffix for the boolean expression.

Equijoin: This departs from Codd's original by defining it to project away redundant duplicate attributes, in spite of which we also have …

Natural join: This "has the nice property that the result is guaranteed not to have two fields with same name". The text mentions that cross-product is a special case.

Division: This is basically Codd's original operator, using the syntax *A*/*B*.

Relational completeness is briefly discussed in Section 4.4 (though defined in terms of the algebra, not the calculus).

There is no material on extension, summarization, semijoin, antijoin, aggregation, or relations of degree zero.

15. E.F. Codd: ***The Relational Model for Database Management, Version 2*** (Addison-Wesley, 1991). 538 pages. Notes by HD.

PRTV gets a mention in **References**, which appears to be little more than a bibliography (and a curiously selective one at that). PRTV's appearance in 1972 is noted as "having preceded all other systems based on the relational model" (a claim that might be open to dispute but is certainly not far off the mark).

Relational operators are described in Chapter 4, **The Basic Operators**, and Chapter 5, **The Advanced Operators**.

In Chapter 4 **Cartesian product** is mentioned as an operator the DBMS "must not support as an explicitly separate operator". The other operators mentioned in this chapter are **project**, **theta-select** ("originally called **theta-restrict**"), **extended theta-select** (allowing the use of logical connectives), **theta-join** (including **equi-join**), **extended theta-join** (allowing the use of logical connectives), **natural join, union, intersection, difference, relational division** (Codd's divide, of course).

So, still no extension or summarization. And no renaming, but in this connection the book contains a bizarre example illustrating the use of equi-join to "join a relation to itself". The columns of the EMP table include ENAME (employee's name) and MGR# (employee number of manager), but no MGR_NAME column for the manager's name, of course. Here's Codd's proposed join to obtain the names of employees' managers:

```
(EMP[EMP# = MGR#]EMP)[EMP#, ENAME, CITY, MGR#, MGR_NAME]
```

[Boldface added]. Notice that the join operands are the same relation (sorry, table). Codd's explanation of theta-join says that the result is "an R-table[1] that contains rows of one operand (say S) concatenated with rows of the second operand (say T), but only where the specified condition is found to hold true. … The condition … involves comparing each value from a column of S with each value from a column of T." So I assume that the left operand of "=" refers to a column of the left-hand EMP and the right to a column on the right. In that case the result must have two attributes named EMP#, two named ENAME, and so on. Which of those two EMP#s, of those two ENAMEs, and so on are the ones included in the projection? And where did the name MGR_NAME come from?

[1] This term is defined as equivalent to relation, on page 17 in a chapter titled "Tables versus Relations".

References

[1] Hugh Darwen: "Why Are There No Relational DBMSs?", available in PDF at www.thethirdmanifesto.com.

[2] C.J. Date: *Relational Database Writings, 1985-1989.* Boston, Mass.: Addison-Wesley (1990).

[3] C.J. Date: "Codd's First Relational Papers: A Critical Analysis", available in PDF at www.thethirdmanifesto.com.

[4] C.J. Date and Hugh Darwen: Appendix A, "A New Relational Algebra", in *Databases, Types and The Relational Model: The Third Manifesto.* 3rd edition, Addison-Wesley (2007).

The book is available in PDF as a free download at www.thethirdmanifesto.com.

[5] Patrick Hall: "Optimisation of a Single Relational Expression in a Relational Database System", *IBM Journal of Research and Development 20,* No 3 (May 1976).

[6] Patrick Hall, Peter Hitchcock, and Stephen Todd: "An Algebra of Relations for Machine Computation", Conf. Record of the 2nd ACM Symposium on Principles of Programming Languages, Palo Alto, California (January 1975).

A copy of this paper, with annotations by Hugh Darwen in 2014, is available in PDF at www.thethirdmanifesto.com.

[7] Stephen Todd: "The Peterlee Relational Test Vehicle—A System Overview", *IBM Systems Journal* **15**, No. 4 (1976).