A Relational Model of Data for Large Shared Data Banks

E. F. CODD

IBM Research Laboratory, San Jose, California

Future users of large data banks must be protected from having to know how the data is organized in the machine (the internal representation). A prompting service which supplies such information is not a satisfactory solution. Activities of users at terminals and most application programs should remain unaffected when the internal representation of data is changed and even when some aspects of the external representation are changed. Changes in data representation will often be needed as a result of changes in query, update, and report traffic and natural growth in the types of stored information.

Existing noninferential, formatted data systems provide users with tree-structured files or slightly more general network models of the data. In Section 1, inadequacies of these models are discussed. A model based on *n*-ary relations, a normal form for data base relations, and the concept of a universal data sublanguage are introduced. In Section 2, certain operations on relations (other than logical inference) are discussed and applied to the problems of redundancy and consistency in the user's model.

KEY WORDS AND PHRASES: data bank, data base, data structure, data organization, hierarchies of data, networks of data, relations,

大規模共用データバンクのための、データのリレーショナルモデル

E.F.コッド

IBM リサーチラボラトリ,サン・ノゼ,カリフォルニア

大規模データバンクの未来のユーザは、機械の中でデータがどのように編成されているかを（すなわち内部表現を）知る責務から保護されなければならない。そうした情報を提供する利用者支援サービスは、満足のいく解決策ではない。端末でのユーザの活動と大部分のアプリケーションプログラムは、データの内部表現が変更された時にも、さらには、外部表現がいくつかの面で変更された時にも、影響を受けずに済むべきである。データ表現に関する変更は、問い合わせ、更新、レポートトラフィックに関する変化と、蓄積される情報のタイプが自然に増えることの結果として、しばしば必要となるだろう。

既存の非推論的な定型データシステムは、ツリー構造のファイル、あるいはそれよりいくらか一般化されたネットワーク型のデータのモデルをユーザに提供する。セクション１では、これらのモデルが不適切であることが論じられる。 *n* 項のリレーションにもとづくモデル、データベースのリレーションについての正規形、汎用的なデータ副言語の概念が紹介される。セクション２では、リレーションを対象とする若干の操作（論理推論を除く）が論じられ、ユーザのモデルにおける冗長性と一貫性の問題に適用される。

キーワードとキーフレーズ:データバンク、データベース、データ構造、データ編成、データの階層、データのネットワーク、リレーション、導出可能

derivability, redundancy, consistency, composition, join, retrieval language, predicate calculus, security, data integrity
CR CATEGORIES: 3.70, 3.73, 3.75, 4.20, 4.22, 4.29

1. Relational Model and Normal Form

1.1. INTRODUCTION

This paper is concerned with the application of elementary relation theory to systems which provide shared access to large banks of formatted data. Except for a paper by Childs [1], the principal application of relations to data systems has been to deductive question-answering systems. Levein and Maron [2] provide numerous references to work in this area.

In contrast, the problems treated here are those of data independence--the independence of application programs and terminal activities from growth in data types and changes in data representation--and certain kinds of data inconsistency which are expected to become troublesome even in nondeductive systems.

The relational view (or model) of data described in Section 1 appears to be superior in several respects to the graph or network model [3, 4] presently in vogue for noninferential systems. It provides a means of describing data with its natural structure only--that is, without superimposing any additional structure for machine representation purposes. Accordingly, it provides a basis for a high level data language which will yield maximal independence between programs on the one hand and machine representation and organization of data on the other.

A further advantage of the relational view is that it forms a sound basis for treating

性、冗長性、一貫性、合成、結合、データ取得言語、述語計算、セキュリティ、データの整合性

CR カテゴリ：3.70, 3.73, 3.75, 4.20, 4.22, 4.29

1. リレーショナルモデルと正規形

1.1. はじめに

この論文は、定型データの大規模な貯蔵庫(bank)に対する共用アクセスを提供するシステムに対する、初歩的なリレーション理論の適用にかかわるものである。Childs[1]による論文を例外として、データシステムに対するリレーションの適用は、主として、推論的問い合わせ-回答システムを対象とするものであった。Levein and Maron [2]は、この分野における研究に関するおびただしい論及を提供している。

それとは対照的に、本論文で扱われるのは、データ独立性、すなわち、データタイプの増加とデータ表現の変更からの、アプリケーションプログラムと端末使用活動の独立性に関する諸問題であり、また、非推論的なシステムにおいてさえ厄介事となることが予想される、ある種のデータ不整合に関する諸問題である。

セクション１で説明されるデータのリレーショナルな見方（あるいはモデル）は、いくつかの点で、現在 非推論的システムの分野で流行しているグラフ型あるいはネットワーク型のモデル[3,4]より優れているように思われる。リレーショナルモデルは、データの自然な構造だけをもちいて、つまり、計算機上の表現のための構造をいっさい付け加えることなくデータを記述する手段を提供する。その結果として、リレーショナルモデルは、プログラムの側と計算機上のデータ表現およびデータ編成の側の間に最大限の独立性をもたらす高水準のデータ言語にとっての基礎を提供するのである。

リレーショナルな見方のもうひとつの利点は、リレーションの導出可能性、冗長性、一貫性を取

derivability, redundancy, and consistency of relations--these are discussed in Section 2. The network model, on the other hand, has spawned a number of confusions, not the least of which is mistaking the derivation of connections for the derivation of relations (see remarks in Section 2 on the "connection trap").

Finally, the relational view permits a clearer evaluation of the scope and logical limitations of present formatted data systems, and also the relative merits (from a logical standpoint) of competing representations of data within a single system. Examples of this clearer perspective are cited in various parts of this paper. Implementations of systems to support the relational model are not discussed.

1.2. DATA DEPENDENCIES IN PRESENT SYSTEMS

The provision of data description tables in recently developed information systems represents a major advance toward the goal of data independence [5, 6, 7]. Such tables facilitate changing certain characteristics of the data representation stored in a data bank. However, the variety of data representation characteristics which can be changed without logically impairing some application programs is still quite limited. Further, the model of data with which users interact is still cluttered with representational properties, particularly in regard to the representation of collections of data (as opposed to individual items). Three of the principal kinds of data dependencies which still need to be removed are: ordering dependence, indexing dependence, and access path dependence. In some systems these dependencies are not clearly separable from one another.

り扱うための堅固な基礎をそれが形成することである。これらは、セクション２で論じられる。他方、ネットワーク型モデルは、数多くの混乱を生みだしてきた。結合の導出をリレーションの導出と取り違えることは、そうした混乱の重要な例である（セクション２の「結合のわな」に関する注釈を見よ）。

最後に、リレーショナルな見方によるならば、既存の定型データシステムの対象範囲と論理的限界、そして、単一のシステム内で競合する複数のデータ表現の（論理的な見地からの）相対的利点も、これまでより明瞭に評価することができる。この、より明瞭な視野の例は、本論文の様々な箇所で引き合いに出される。リレーショナルモデルをサポートするシステムの実装については論じられない。

1.2. 既製システムにおけるデータ依存性

最近開発された情報システムにおいてデータ記述表が提供されていることは、データ独立というゴールに向けての大きな前進を意味している[5,6,7]。こうした表は、データバンクに保管されるデータの表現に係るある種の特性を変更することを容易にする。とはいえ、アプリケーションプログラムのいくつかが論理的に正しく機能しなくなる事態を招かずに変更できるデータ表現上の特性の種類は、いまだ極めて限られている。さらに言えば、ユーザが接するデータのモデルには、いまなお、データ表現上の属性、とりわけ（個々のデータと対比される）データの集まりの表現に関する属性が取り散らかっている。今後さらに除去されるべき主たるデータ依存性を３種類挙げれば：順序付け依存、索引付け依存、そしてアクセス経路依存、である。いくつかのシステムでは、これらの依存性はお互いに明瞭に区別できない。

訳者注解　本論文で、「表現(representation)」という言葉がデータに対して用いられる場合、データの本質でない、状況に依存した表れ方というニ

1.2.1. *Ordering Dependence*. Elements of data in a data bank may be stored in a variety of ways, some involving no concern for ordering, some permitting each element to participate in one ordering only, others permitting each element to participate in several orderings. Let us consider those existing systems which either require or permit data elements to be stored in at least one total ordering which is closely associated with the hardware-determined ordering of addresses. For example, the records of a file concerning parts might be stored in ascending order by part serial number. Such systems normally permit application programs to assume that the order of presentation of records from such a file is identical to (or is a subordering of) the stored ordering. Those application programs which take advantage of the stored ordering of a file are likely to fail to operate correctly if for some reason it becomes necessary to replace that ordering by a different one. Similar remarks hold for a stored ordering implemented by means of pointers.

It is unnecessary to single out any system as an example, because all the well-known information systems that are marketed today fail to make a clear distinction between order of presentation on the one hand and stored ordering on the other. Significant implementation problems must be solved to provide this kind of independence.

1.2.2. *Indexing Dependence*. In the context of formatted data, an index is usually thought of as a purely performance-oriented component of the data representation. It tends to improve response to queries and updates and, at the

ュアンスが強く感じられる。

1.2.1. *順序付け依存*. データバンクに含まれるデータの諸要素は、様々な形式で保管される可能性がある。順序付けにこだわらない形式もあれば、個々の要素がひとつの順序付けのみに加わることを許す形式もあり、また、数種の順序付けに加わることを許すものもある。データ要素が、少なくともひとつの、ハードウェアで決定されるアドレスの順序に強く結びついた全順序[①]に従って保管されていることを要求あるいは許容する既製システムを考えてみよう。例えば、部品に関するファイルのレコードは、部品連番の昇順に並べて保管されているかもしれない。通常、このようなシステムは、そうしたファイルに含まれるレコードの提示順が、保管順序と同一である（か、その部分順序である）と、アプリケーションプログラムが仮定することを許容する。保管順序を異なるものと取り換えることが、何らかの理由で必要となった場合、その順序を利用するアプリケーションプログラムは正しく動作しなくなる可能性が高い。ポインタを用いて実装された保管順序についても同様のことが言える。

なんらかのシステムをとりたててひとつ例示する必要はない。今日市販されている名の通った情報システムはすべて、提示の順序と保管順序を明確に区別することに失敗しているからである。この種の独立性を提供するために、実装上の多大の問題が解決されなければならない。

1.2.2. *索引付け依存*. 定型データの文脈では、通常、索引は純粋に性能志向のデータ表現構成要素と考えられている。索引を用いれば、問い合わせと更新のレスポンスは改善し、それと同時に挿入と削除のレスポンス速度は落ちる傾向があ

①訳注：全順序とは要素のすべてについて先後が定まるような順序関係。家系図のように先後が決まらない組合せを許す順序を半順序と呼ぶのに対する用語。

same time, slow down response to insertions and deletions. From an informational standpoint, an index is a redundant component of the data representation. If a system uses indices at all and if it is to perform well in an environment with changing patterns of activity on the data bank, an ability to create and destroy indices from time to time will probably be necessary. The question then arises: Can application programs and terminal activities remain invariant as indices come and go?

Present formatted data systems take widely different approaches to indexing. TDMS [7] unconditionally provides indexing on all attributes. The presently released version of IMS [5] provides the user with a choice for each file: a choice between no indexing at all (the hierarchic sequential organization) or indexing on the primary key only (the hierarchic indexed sequential organization). In neither case is the user's application logic dependent on the existence of the unconditionally provided indices. IDS [8], however, permits the file designers to select attributes to be indexed and to incorporate indices into the file structure by means of additional chains. Application programs taking advantage of the performance benefit of these indexing chains must refer to those chains by name. Such programs do not operate correctly if these chains are later removed.

1.2.3. *Access Path Dependence*. Many of the existing formatted data systems provide users with tree-structured files or slightly more general network models of the data. Application programs developed to work with these systems tend to be logically impaired if the trees or networks are changed in structure. A simple example follows.

る。情報という観点からは、索引は冗長なデータ表現構成要素である。あるシステムが仮にも索引を使い、かつ、データバンクでの活動パターンが変化する環境で性能よく動作することが要求されるのであれば、折に触れて索引を作成し破棄する能力が、たぶん必要となるだろう。そこで、疑問が生じる：アプリケーションプログラムと端末使用活動は、索引が作成され、また、消え去るに際して、不変のままとどまることができるだろうか？

既製の定型データシステムは、索引付けに対して実にさまざまのアプローチを採る。TDMS[7]は、すべての属性に対して無条件に索引付けを行う。IMS の現在リリースされているバージョン[5]は、ファイルごとの選択をユーザに許す：索引付けをまったくしないか（階層順編成）、あるいはプライマリキーのみに索引付けするか（階層索引付き順編成）の選択である。いずれのケースでも、無条件に提供される索引の存在にユーザのアプリケーションロジックが依存することはない。しかしながら、IDS[8]は、索引付けする（複数の）属性を選択し、付加的チェーンという手法を用いて索引をファイル構造に組み込むことを、ファイル設計者に許している。こうした索引チェーンによる性能上の恩恵を受けるには、アプリケーションプログラムは、名前によってこれらのチェーンを参照しなければならない。そのようなプログラムは、これらのチェーンが後になって除去された場合、正しく動作しなくなるのである。

1.2.3. *アクセス経路依存*. 既製の定型データシステムの多くは、ツリー構造のファイル、あるいはそれよりいくらか一般化されたネットワーク型のデータのモデルをユーザに提供する。これらのシステムと組み合わせて動作するよう開発されたアプリケーションプログラムは、ツリーやネットワークの構造が変更された場合、論理的に正しく機能しなくなる場合が多い。

Suppose the data bank contains information about parts and projects. For each part, the part number, part name, part description, quantity-on-hand, and quantity-on-order are recorded. For each project, the project number, project name, project description are recorded. Whenever a project makes use of a certain part, the quantity of that part committed to the given project is also recorded. Suppose that the system requires the user or file designer to declare or define the data in terms of tree structures. Then, any one of the hierarchical structures may be adopted for the information mentioned above (see Structures 1-5).

部品とプロジェクトに関する情報を含んだデータバンクを想定しよう。部品各々について、部品番号、部品名、部品説明、在庫数量、発注済数量が記録される。プロジェクト各々について、プロジェクト番号、プロジェクト名、プロジェクト説明が記録される。プロジェクトで部品を使う場合には常に、そのプロジェクトに支給されたその部品の数量も記録される。このシステムでは、ユーザあるいはファイル設計者は、データをツリー構造として定義あるいは宣言することを要求されるとしよう。このとき、上述した情報に対して、以下の階層構造のいずれを採用することもできる（構造 1-5 を参照）。

Structure 1. Projects Subordinate to Parts

File	*Segment*	*Fields*
F	PART	part #
		part name
		part description
		quantity-on-hand
		quantity-on-order
	PROJECT	project #
		project name
		project description
		quantity committed

構造 1. プロジェクトが部品に従属

ファイル	*セグメント*	*フィールド*
F	部品	部品番号
		部品名
		部品説明
		在庫数量
		発注済数量
	プロジェクト	プロジェクト番号
		プロジェクト名
		プロジェクト説明
		支給済数量

Structure 2. Parts Subordinate to Projects

File	*Segment*	*Fields*
F	PROJECT	project #
		project name
		project description
	PART	part #
		part name
		part description
		quantity-on-hand
		quantity-on-order
		quantity committed

構造 2. 部品がプロジェクトに従属

ファイル	*セグメント*	*フィールド*
F	プロジェクト	プロジェクト番号
		プロジェクト名
		プロジェクト説明
	部品	部品番号
		部品名
		部品説明
		在庫数量
		発注済数量
		支給済数量

Structure 3. Parts and Projects as Peers Commitment Relationship Subordinate to Projects

File	*Segment*	*Fields*
F	PART	part #
		part name
		part description
		quantity-on-hand
		quantity-on-order
G	PROJECT	project #
		project name
		project description
	PART	part #
		quantity committed

構造 3. 部品とプロジェクトが同格
支給関係がプロジェクトに従属

ファイル	*セグメント*	*フィールド*
F	部品	部品番号
		部品名
		部品説明
		在庫数量
		発注済数量
G	プロジェクト	プロジェクト番号
		プロジェクト名
		プロジェクト説明
	部品	部品番号
		支給済数量

Structure 4. Parts and Projects as Peers Commitment Relationship Subordinate to Parts

File	*Segment*	*Fields*
F	PART	part #
		part name
		part description
		quantity-on-hand
		quantity-on-order
	PROJECT	project #
		quantity committed
G	PROJECT	project #
		project name
		project description

構造 4. 部品とプロジェクトが同格
支給関係が部品に従属

ファイル	*セグメント*	*フィールド*
F	部品	部品番号
		部品名
		部品説明
		在庫数量
		発注済数量
	プロジェクト	プロジェクト番号
		支給済数量
G	プロジェクト	プロジェクト番号
		プロジェクト名
		プロジェクト説明

Structure 5. Parts, Projects, and Commitment Relationship as Peers

File	*Segment*	*Fields*
F	PART	part #
		part name
		part description
		quantity-on-hand
		quantity-on-order
G	PROJECT	project #
		project name
		project description
H	COMMIT	part #
		project #
		quantity committed

構造 5. 部品・プロジェクト・支給関係が同格

ファイル	*セグメント*	*フィールド*
F	部品	部品番号
		部品名
		部品説明
		在庫数量
		発注済数量
G	プロジェクト	プロジェクト番号
		プロジェクト名
		プロジェクト説明
H	支給	部品番号
		プロジェクト番号
		支給済数量

Now, consider the problem of printing out the part number, part name, and quantity committed for every part used in the project whose project name is "alpha." The following observations may be made regardless of which available tree-oriented information system is selected to tackle this problem. If a program P is developed for this problem assuming one of the five structures above—that is, P makes no test to determine which structure is in effect-then P will fail on at least three of the remaining structures. More specifically, if P succeeds with structure 5, it will fail with all the others; if P succeeds with structure 3 or 4, it will fail with at least 1, 2, and 5; if P succeeds with 1 or 2, it will fail with at least 3, 4, and 5. The reason is simple in each case. In the absence of a test to determine which structure is in effect, P fails because an attempt is made to execute a reference to a nonexistent file (available systems treat this as an error) or no attempt is made to execute a reference to a file containing needed information. The reader who is not convinced should develop sample programs for this simple problem.

Since, in general, it is not practical to develop application programs which test for all tree structurings permitted by the system, these programs fail when a change in structure becomes necessary.

Systems which provide users with a network model of the data run into similar difficulties. In both the tree and network cases, the user (or his program) is required to exploit a collection of user access paths to the data. It does not matter whether these paths are in close correspondence with pointer-defined paths in the stored representation—in IDS the correspondence is extremely simple, in TDMS it is just the

さて、”alpha”というプロジェクト名のプロジェクトで使用されたすべての部品の部品番号、部品名、支給済数量を印刷するという問題を考えてみよう。利用可能なツリー指向型情報システムのいずれがこの問題に取り組むために選ばれたとしても、以下の所見が得られるだろう。上記５つの構造のうちひとつを前提としてこの問題を解くようプログラムPが開発されたとすればーすなわち、Pは、どの構造が実際に用いられているか判断するテストをいっさい行わないとすればーPは残りの構造のうち少なくとも３つで失敗するだろう。より具体的に言えば、Pが構造５をうまく扱えるとすれば、その他全てにおいて失敗するだろう。Pが構造３または４をうまく扱えるなら、少なくとも構造1,2と５で失敗する。Pが１または２をうまく扱えるなら、少なくとも構造3,4と５で失敗する。それぞれのケースで、理由は単純である。どの構造が実際に用いられているかを判断するテストを行わないため、実際には存在しないファイルを参照しようするか（利用可能なシステムはこれをエラーとして取り扱う）、必要な情報を含むファイルに対する参照を実行しようとしないために、Pは失敗する。納得できない読者はこの単純な問題用にサンプルプログラムを開発すべきである。

一般的に言って、システムが許すツリーの組み方すべてを試すアプリケーションプログラムを開発するのは実際的ではないので、構造に変更が必要となったとき、これらのプログラムはうまく動かなくなるのである。

ユーザにネットワーク型モデルを提供するシステムも、同様の困難に陥る。ツリー型、ネットワーク型いずれでも、ユーザ（またはユーザのプログラム）は、ユーザ向けのデータアクセス経路一式を利用することを要求される。こうした経路が、保管形式上でポインタにより定義された経路と密接に対応するのかどうかは—IDSでは対応関係は極めて単純であり、TDMSではまったくその逆であるー問題ではない。保管形式の如何によら

opposite. The consequence, regardless of the stored representation, is that terminal activities and programs become dependent on the continued existence of the user access paths.

One solution to this is to adopt the policy that once a user access path is defined it will not be made obsolete until all application programs using that path have become obsolete. Such a policy is not practical, because the number of access paths in the total model for the community of users of a data bank would eventually become excessively large.

1.3. A RELATIONAL VIEW OF DATA

The term relation is used here in its accepted mathematical sense. Given sets S_1, S_2, … , S_n (not necessarily distinct), R is a relation on these n sets if it is a set of n-tuples each of which has its first element from S_1, its second element from S_2, and so on. [1] We shall refer to S_j as the jth *domain* of R. As defined above, R is said to have *degree n*. Relations of degree 1 are often called *unary*, degree 2 *binary*, degree 3 *ternary*, and degree n *n-ary*.

[1] More concisely, R is a subset of the Cartesian product $S_1 \times S_2 \times \ldots \times S_n$

.

For expository reasons, we shall frequently make use of an array representation of relations, but it must be remembered that this particular representation is not an essential part of the relational view being expounded. An array which represents an n-ary relation R has the following properties:

ず、結果として、それらのユーザアクセス経路が存在し続けることに、端末使用活動とプログラムが依存するようになるのである。

これに対するひとつの解決策は、ユーザアクセス経路を一度定義したならば、その経路を使用するアプリケーションがすべて廃棄されるまでその経路を廃棄しないという方針を採用することである。そうした方針は実際的ではない。ひとつのデータバンクのユーザコミュニティが使用するモデル全体に含まれるアクセス経路が、時の経過につれ膨大な数となってしまうからである。

1.3. データのリレーショナルな見方

リレーションという用語は、ここでは、一般に受け入れられた数学的意味で用いられる。集合 S_1, S_2, … , S_n（必ずしも各々別個でなくてもよい）を与えられたとき、R が n-タプルの集合であって、各タプルの最初の要素が S_1 から採られ、2 番目の要素が S_2 から採られ、以下もそれに続く場合に、R はこれら n 個の集合上のリレーションである [1]。S_j を R の j 番目の *ドメイン*と呼ぶことにする。上記のように定義された場合、R は*次数 n* を持つと言う。次数 1 のリレーションは*単項*としばしば呼ばれ、次数 2 は *2 項*、次数 3 は *3 項*、そして、次数 n は n *項*と呼ばれる。

[1] より正確には、R はデカルト積②$S_1 \times S_2 \times \ldots \times S_n$ の部分集合である。

説明をわかりやすくする目的で、私たちはリレーションの配列表現を頻繁に用いることにするが、この特定の表現が、解説対象であるリレーショナルな見方の本質的な部分ではないということは、心に留めておかなければならない。n 次のリレーションを表現する配列は次の特性を持つ：

(1) 各行は R の n-タプルひとつを表す。

② 訳注：「デカルト積」は「直積」と同じ意味である。

(1) Each row represents an n-tuple of R.
(2) The ordering of rows is immaterial.
(3) All rows are distinct.
(4) The ordering of columns is significant--it corresponds to the ordering $S_1, S_2, \cdots, S_n$ of the domains on which R is defined (see, however, remarks below on domain-ordered and domain-unordered relations).
(5) The significance of each column is partially conveyed by labeling it with the name of the corresponding domain.

The example in Figure 1 illustrates a relation of degree 4, called *supply*, which reflects the shipments-in-progress of parts from specified suppliers to specified projects in specified quantities.

supply	*(supplier*	*part*	*project*	*quantity)*
	1	2	5	17
	1	3	5	23
	2	3	7	9
	2	7	5	4
	4	1	1	12

FIG. 1. A relation of degree 4

One might ask: If the columns are labeled by the name of corresponding domains, why should the ordering of columns matter? As the example in Figure 2 shows, two columns may have identical headings (indicating identical domains) but possess distinct meanings with respect to the relation. The relation depicted is called component. It is a ternary relation, whose first two domains are called part and third domain is called quantity. The meaning of component (x, y, z) is that part x is an immediate component (or subassembly) of part y, and z units of part x are needed to assemble one unit

(2) 行の並び順は重要では無い。
(3) すべての行は互いに異なる。
(4) 列の並び順には意味がある。--並び順は、その上に R が定義されているドメイン群 $S_1, S_2, \cdots, S_n$ の並び順に対応している（ただし、ドメイン順序ありとドメイン順序なしのリレーションに関する下記の注釈を参照せよ）。
(5) 各列の意味は、対応するドメインの名前を列に付すことにより、ある程度伝えられる。

図 1 の例は、4 次のリレーションを図解したものである。このリレーションは「*供給*」と呼ばれ、指定された供給業者から指定されたプロジェクトに対する、指定された数量の部品の出荷のうち仕掛かり中のものを描写している。

供給	*(供給業者*	*部品*	プロジェクト	*数量)*
	1	2	5	17
	1	3	5	23
	2	3	7	9
	2	7	5	4
	4	1	1	12

図 1. 次数 4 のリレーションの例

和訳省略　列の並び順が重要である理由を、部品構成―（部品, 部品, 数量）の例を用いて説明。最初の部品は子部品、二番目は親部品。両者ともドメインは同一なので、並び順で両者を区別するという。

of part y. It is a relation which plays a critical role in the parts explosion problem.

component	*(part*	*part*	*quantity)*
	1	5	9
	2	5	7
	3	5	2
	2	6	12
	3	6	3
	4	7	1
	6	7	1

FIG. 2. A relation with two identical domains

It is a remarkable fact that several existing information systems (chiefly those based on tree-structured files) fail to provide data representations for relations which have two or more identical domains. The present version of IMS/360 [5] is an example of such a system.

The totality of data in a data bank may be viewed as a collection of time-varying relations. These relations are of assorted degrees. As time progresses, each *n*-ary relation may be subject to insertion of additional *n*-tuples, deletion of existing ones, and alteration of components of any of its existing *n*-tuples.

In many commercial, governmental, and scientific data banks, however, some of the relations are of quite high degree (a degree of 30 is not at all uncommon). Users should not normally be burdened with remembering the domain ordering of any relation (for example, the ordering supplier, then part, then project, then quantity in the relation supply).

Accordingly, we propose that users deal, not with relations which are domain-ordered, but with relationships which are their domain-unordered counterparts. [2] To accomplish this, domains must be uniquely identifiable at least within any given relation, without using

既存の情報システムのいくつか（主にツリー構造のファイルに基づくもの）が、同一ドメインを2つあるいはそれ以上含むリレーションのデータ表現をうまく提供できないというのは、注目に値する事実である。IMS/360 の現行バージョン[5]は、そうしたシステムの一例である。

データバンク中のデータ全体は、時の経過に伴い変化するリレーションの集まりとみることができる。これらのリレーションの次数はさまざまである。時の経過につれ、各 *n* 次リレーションに対して、*n*-タプルがさらに挿入され、既存のものが削除され、既存の *n*-タプルの構成要素が変更される。

和訳省略　ドメイン順序なしのリレーションというアイデアの導入。

部品構成―（部品，部品，数量）

と書く代わりに、役割という概念を導入し、

部品構成―（下位.部品，上位.部品，数量）

と書く。これで、ドメインの順序は不要となる。

今日ではこちらが主流。

コッド博士はこれを「リレーションシップ」と名付ける。ＥＲモデルの「リレーションシップ」とは異なるので注意。

以下の議論でコッド博士は、「ユーザ向けのリレーショナルモデル（＝論理モデル）」に対してリレーションシップを用い、「保管形式（＝物理モデル）」に対してリレーションを用いる。

position. Thus, where there

とはいえ、多くの議論は両者に当てはまる。

are two or more identical domains, we require in each case that the domain name be qualified by a distinctive role name, which serves to identify the role played by that domain in the given relation. For example, in the relation component of Figure 2, the first domain part might be qualified by the role name sub, and the second by super, so that users could deal with the relationship component and its domains--sub.part super.part, quantity--without regard to any ordering between these domains.

To sum up, it is proposed that most users should interact with a relational model of the data consisting of a collection of time-varying relationships (rather than relations). Each user need not know more about any relationship than its name together with the names of its domains (role qualified whenever necessary).[3] Even this information might be offered in menu style by the system (subject to security and privacy constraints) upon request by the user.

There are usually many alternative ways in which a relational model may be established for a data bank. In order to discuss a preferred way (or normal form), we must first introduce a few additional concepts (active domain, primary key, foreign key, nonsimple domain) and establish some links with terminology currently in use in information systems programming. In the remainder of this paper, we shall not bother to distinguish between relations and relationships except where it appears advantageous to be explicit.

[2] In mathematical terms, a relationship is an equivalence class of those relations that are equivalent under permutation of domains (see Section 2.1.1).

[3] Naturally, as with any data put into and retrieved from a computer system, the user will normally make far more effective use of the data if he is aware of its meaning.

Consider an example of a data bank which includes relations concerning parts, projects, and suppliers. One relation called part is defined on the following domains:

(1) part number
(2) part name
(3) part color
(4) part weight
(5) quantity on hand
(6) quantity on order

and possibly other domains as well. Each of these domains is, in effect, a pool of values, some or all of which may be represented in the data bank at any instant. While it is conceivable that, at some instant, all part colors are present, it is unlikely that all possible part weights, part names, and part numbers are. We shall call the set of values represented at some instant the active domain at that instant.

和訳省略　「アクティブドメイン」という概念の説明。
アクティブドメインは、ドメインの要素のうち実際にリレーションで用いられているものの集合。例えば部品番号が８桁の文字列として定義されているとすれば、部品番号のドメインは、８桁の文字列すべての集合であり、アクティブドメインは、そのうち登録済みの部品番号、すなわち実際に部品番号として「部品リレーション」で使用されているものの集合である。
ただし、論文の残りの部分ではアクティブドメインは一か所でしか用いられていない。

Normally, one domain (or combination of domains) of a given relation has values which uniquely identify each element (n-tuple) of that relation. Such a domain (or combination) is called a primary key. In the example above, part number would be a primary key, while part color would not be. A primary key is nonredundant if it is either a simple domain (not a combination) or a combination such that none of the participating simple domains is superfluous in uniquely identifying each element. A relation may possess more than one nonredundant primary key. This would be the case in the example if different parts were always given distinct names. Whenever a relation has two or

和訳省略　「プライマリキー」の定義。ここで言うプライマリキーは、現代の用語でいう「スーパーキー」であり、「非冗長なプライマリキー」は現代なら「候補キー」である。現代の用語でいうプライマリキーを、コッド博士は「ザ・プライマリキー」と呼んでいる。ただし論文の残りの部分では「ザ」付きのプライマリキーは、一か所を除き、出てこない。博士は現代で言う候補キーとプライマリキーの区別にさほど拘らなかったようである。
余談だが、現代の実務上は、一般にプライマリキーは変更不可、他の候補キーは変更可と捉える場合も多い。その意味では両者の区別は重要と思う。こうしたキーの変更可否の概念を明快に説明している文献はあるのだろうか。

more nonredundant primary keys, one of them is arbitrarily selected and called the primary key of that relation.

A common requirement is for elements of a relation to cross-reference other elements of the same relation or elements of a different relation. Keys provide a user-oriented means (but not the only means) of expressing such cross-references. We shall call a domain (or domain combination) of relation R a *foreign key* if it is not the primary key of R but its elements are values of the primary key of some relation S (the possibility that S and R are identical is not excluded). In the relation supply of Figure 1, the combination of supplier, part, project is the primary key, while each of these three domains taken separately is a foreign key.

和訳省略　「外部キー」の定義。プライマリキーは外部キーになれないという、本来不要な制約が設けられている（この論文執筆時点ではプライマリキーを外部キーとする必要性に考えが及ばなかったか）。その他は、現代の用語でいう外部キーと同じ。

In previous work there has been a strong tendency to treat the data in a data bank as consisting of two parts, one part consisting of entity descriptions (for example, descriptions of suppliers) and the other part consisting of relations between the various entities or types of entities (for example, the supply relation). This distinction is difficult to maintain when one may have foreign keys in any relation whatsoever. In the user's relational model there appears to be no advantage to making such a distinction (there may be some advantage, however, when one applies relational concepts to machine representations of the user's set of relationships).

先行研究においては、データバンク中のデータを二つの部分から構成されるものとして取り扱う強い傾向が存在した。すなわち、ひとつ目の部分はエンティティの記述項目（例えば、供給業者に関する記述項目）から成り、他の部分はさまざまなエンティティやエンティティタイプの間の関係（例えば、供給関係）から成るとするのである。任意のリレーションに、どんな外部キーを設けてもよいとするなら、この区別を維持することは困難となる。ユーザ向けのリレーショナルモデルでは、こうした区別を設けることに利点は無いように見える（とはいえ、ユーザ向けのリレーションシップの集合に対する計算機上の表現にリレーショナル概念を適用する際には、なんらかの利点があるかもしれない）。

訳者注解　従来のツリー型・ネットワーク型のモデルに対して、すでに 3 つの依存性を挙げて、実務的な観点からの問題を指摘したが、ここでは「そもそも論」として、従来モデルのコンセプトのまずさを指摘している、すなわち従来モデルに特徴的な「属性はフィールドで、関連はポインタで表現する」という二分法は恣意的であり貫徹できな

いとの指摘と訳者は理解する。そうした観点からすると、後に提案されたチェン博士の ER モデルは、せっかく否定した属性と関連の二分法を今一度裏口から持ち込むようなものであり、コッド博士にとっては承服しがたい面があっただろう。ER モデル論文の掲載にコッド博士が反対したことを、チェン博士自身が記録している。
http://bit.csc.lsu.edu/~chen/chen.html

訳者余談　リレーショナルモデルは、「ドメイン」だけを材料とする一元的モデルだが、ER モデルは「実体集合」「値集合」という 2 種類の材料を持つ二元的モデルであって、数学的には全く異なる。ER モデルにおける「実体集合」「値集合」の区別の恣意性のゆえに(*1)、実態と関連、属性と関連の区別も不安定になっている。もともと無理な区別なのである。

(*1) 日付は実体か値か―普通は値と考えると思われるが、稼働日テーブルでは日付に属性があるので、実体とみる見方も可能である。

和訳省略　入れ子のリレーションすなわち、リレーションをドメインとするリレーションも可能であるという主張。ただし、ここでいったん入れ子のリレーションを許容するものの、後の部分で正規形（第 1 正規形）を定義することにより、入れ子のリレーションは実質的にリレーショナルモデルの対象外とされる。

本論文に先立つ 1969 年論文では、入れ子のリレーションを除外しなかったために、データ言語のために二階の述語論理が必要とされていた。本論文では、その除外により、一階の述語論理で十分となっている。

So far, we have discussed examples of relations which are defined on simple domains--domains whose elements are atomic (nondecomposable) values. Nonatomic values can be discussed within the relational framework. Thus, some domains may have relations as elements. These relations may, in turn, be defined on nonsimple domains, and so on. For example, one of the domains on which the relation employee is defined might be salary history. An element of the salary history domain is a binary relation defined on the domain date and the domain salary. The salary history domain is the set of all such binary relations. At any instant of time there are as many instances of the salary history relation in the data bank as there are employees. In contrast, there is only one instance of the employee relation.

The terms attribute and repeating group in

present data base terminology are roughly analogous to simple domain and nonsimple domain, respectively. Much of the confusion in present terminology is due to failure to distinguish between type and instance (as in "record") and between components of a user model of the data on the one hand and their machine representation counterparts on the other hand (again, we cite "record" as an example).

1.4. NORMAL FORM

（省略）

1.4. 正規形

和訳省略　第 1 正規形（繰返し項目を含まないリレーション）を定義し、非第 1 正規形からの正規化の手順を示している。内容は周知なので省略。第 2 以上の正規形は本論文では示されないが、その存在を認識していることを示唆する文がある（下記）。

Further operations of a normalizing kind are possible. These are not discussed in this paper.

1.5. SOME LINGUISTIC ASPECTS

The adoption of a relational model of data, as described above, permits the development of a universal data sublanguage based on an applied predicate calculus. A first-order predicate calculus suffices if the collection of relations is in normal form. Such a language would provide a yardstick of linguistic power for all other proposed data languages, and would itself be a strong candidate for embedding (with appropriate syntactic modification) in a variety of host languages (programming, command- or problem-oriented). While it is not the purpose of this paper to describe such a language in detail, its salient features would be as follows.

Let us denote the data sublanguage by *R* and the host language by *H*. *R* permits the declaration of relations and their domains. Each

1.5. いくつかの言語的側面

上述したデータのリレーショナルモデルを採用することによって、述語論理の応用にもとづく汎用的なデータ副言語を開発することが可能になる。リレーションの集まりが正規形であれば、一階述語論理で事足りる。このような言語は、提案された他の全てのデータ言語の言語としての力を計る基準尺を提供し、それ自身、（適当な文法上の修正を加えた上で）さまざまなホスト言語（プログラミング言語、コマンド形式言語、あるいは、問題指向型言語）への埋め込み用として、有力な候補となるだろう。こうした言語を詳細に記述することは本論文の目的ではないが、その顕著な特性は以下のようなものとなるだろう。

データ副言語を *R*、ホスト言語を *H* で示そう。*R* は、リレーション群とそのドメイン群の宣言を許す。リレーションの宣言はそれぞれ、その

declaration of a relation identifies the primary key for that relation. Declared relations are added to the system catalog for use by any members of the user community who have appropriate authorization. *H* permits supporting declarations which indicate, perhaps less permanently, how these relations are represented in storage. *R* permits the specification for retrieval of any subset of data from the data bank. Action on such a retrieval request is subject to security constraints.

The universality of the data sublangnage lies in its descriptive ability (not its computing ability). In a large data bank each subset of the data has a very large number of possible (and sensible) descriptions, even when we assume (as we do) that there is only a finite set of function subroutines to which the system has access for use in qualifying data for retrieval. Thus, the class of qualification expressions which can be used in a set specification must have the descriptive power of the class of well-formed formulas of an applied predicate calculus. It is well known that to preserve this descriptive power it is unnecessary to express (in whatever syntax is chosen) every formula of the selected predicate calculus. For example, just those in prenex normal form are adequate [9].

Arithmetic functions may be needed in the qualification or other parts of retrieval statements. Such functions can be defined in *H* and invoked in *R*.

A set so specified may be fetched for query purposes only, or it may be held for possible

リレーションのプライマリキーを識別する。宣言されたリレーションは、適切な権限を持つユーザコミュニティのメンバなら誰でも使えるようシステムカタログに追加される。*H*は、リレーションが記憶装置上でどのように表現されるかを指示する（たぶんリレーションの宣言ほどには長持ちしない）宣言の支援を可能にする。*R*は、データバンクから、データのどんなサブセット③であれ取得するための仕様記述（訳注：SQL の SELECT 文に相当）を許す。そうしたデータ取得要求に基づくアクションはセキュリティ制約の対象となる。

このデータ副言語の汎用性は、その記述能力に存する（計算能力にではなく）。大規模データバンクにおいては、データのサブセットおのおのは、極めて多数の、あり得る（そして意味のある）記述を含んでいる。取得するデータを限定する際に使用する目的でシステムがアクセスし得る関数サブルーチンの集合が有限でしかないと仮定しても（実際そう仮定するのだが）、それは変わらない。それ故に、ひとつのセット仕様の中で用いることができる限定式のクラスは、述語論理のひとつの応用における整論理式④のクラスに相当する記述力を持たなければならないのである。この記述力を維持するために、選択された述語論理のあらゆる式を（選ばれた文法が何であれ、それに従って）表現する必要はないことはよく知られている。例えば、冠頭標準形⑤に属する式だけで十分用に足る。

限定式や、取得用ステートメントの他の部分で、算術演算関数が必要かもしれない。そうした関数は、*H*で定義し、*R* 内で呼び出すことができる。

セットは、問い合わせ目的に限定して取ってくる(fetch)こともできるし、あり得る変更に備えて

---

③訳注：部分集合。以下、データの集合という場合、集合をセットと訳す。

④訳注：文法に正しく従った論理式。

⑤訳注：「全ての～について」、「ある～について」といった記述（量化子）が式の先頭にまとめて置かれた形の式。ここでコッドが主張しているのは、一階述語論理のすべての式は冠頭標準形に直すことができるため、冠頭標準形だけサポートすれば良いということである。

changes. Insertions take the form of adding new elements to declared relations without regard to any ordering that may be present in their machine representation. Deletions which are effective for the community (as opposed to the individual user or sub-communities) take the form of removing elements from declared relations. Some deletions and updates may be triggered by others, if deletion and update dependencies between specified relations are declared in *R*.

One important effect that the view adopted toward data has on the language used to retrieve it is in the naming of data elements and sets. Some aspects of this have been discussed in the previous section. With the usual network view, users will often be burdened with coining and using more relation names than are absolutely necessary, since names are associated with paths (or path types) rather than with relations.

Once a user is aware that a certain relation is stored, he will expect to be able to exploit [5] it using any combination of its arguments as "knowns" and the remaining arguments as "unknowns," because the information (like Everest) is there. This is a system feature (missing from many current information systems) which we shall call (logically) symmetric exploitation of relations. Naturally, symmetry in performance is not to be expected.

[5] Exploiting a relation includes query, update, and delete.

保持しておくこともできる。挿入は、機械上の表現において存在するかもしれない順序付けに一切気を遣うことなく、宣言されたリレーションに新しい要素を追加する形をとる。コミュニティに対して効力を持つ削除は（個々のユーザやサブコミュニティに効力を持つ削除と違って）、宣言されたリレーションから要素を除去する形をとる。*R*において、特定のリレーション間に削除・更新依存関係が宣言されているならば、削除と更新は、他の削除と更新によって引き起こされる(triggered)かもしれない。

データに対するこうした見方を採用することで、データ取得用言語が受ける重要な影響のひとつは、データの要素とセットの名前付けに関するものである。このことのいくつかの側面については、前のセクションで論じた。通常のネットワーク型の見方を用いた場合、ユーザは、絶対に必要とされる以上のリレーション名を作り、使用するという負担をしばしば強いられる。というのは、リレーションにではなく経路（または経路の型）に名前が結びついているからである。

あるリレーションが保管されていることをユーザが知っていれば、その引数[6]のいずれの組合せでもよいから「既知」とし、残りの引数を「未知」とすることにより、リレーションを探索[5]することができると、ユーザは期待するだろう。情報はそこに、（エベレストのごとく）置かれているからである。これは、（論理的に）対称なリレーション探索と私たちが呼ぶ（多くの現行情報システムは備えていない）システム特性である。当然ながら、性能に関する対称性を期待すべきではない。

[5] リレーションの探索には、問い合わせ、更新、削除が含まれる。

[6]訳注：引数とはドメインのこと。リレーションを述語とみた場合、ドメインは述語の引数となる。例えば「供給する（供給業者, 部品）」をリレーションと見れば、供給業者・部品はドメインであり、述語とみれば、引数である。

To support symmetric exploitation of a single binary relation, two directed paths are needed. For a relation of degree *n*, the number of paths to be named and controlled is *n* factorial.

Again, if a relational view is adopted in which every *n*-ary relation ($n > 2$) has to be expressed by the user as a nested expression involving only binary relations (see Feldman's LEAP System [10], for example) then $2n - 1$ names have to be coined instead of only $n + 1$ with direct *n*-ary notation as described in Section 1.2. For example, the 4-ary relation supply of Figure 1, which entails 5 names in *n*-ary notation, would be represented in the form

P(supplier, *Q*(part, *R*(project, quantity)))

in nested binary notation and, thus, employ 7 names.

A further disadvantage of this kind of expression is its asymmetry. Although this asymmetry does not prohibit symmetric exploitation, it certainly makes some bases of interrogation very awkward for the user to express (consider, for example, a query for those parts and quantities related to certain given projects via *Q* and *R*).

1.6. EXPRESSIBLE, NAMED, AND STORED RELATIONS

Associated with a data bank are two collections of relations: the named set and the expressible set. The named set is the collection of all those relations that the community of

単一の 2 項リレーションで、対称な探索をサポートするには、2 つの有向経路⑦が必要とされる。n 次のリレーションひとつについて、名前を付し管理しなければならない経路の数は n の階乗となる⑧。

さらにまた、あらゆる *n* 項リレーション（$n > 2$）を、2 項リレーションのみを含む入れ子のリレーションとして表現することをユーザに要求するようなリレーショナルな見方（訳注：モデル）を採用したとしよう（例えば、Feldman の LEAP システム[10]を参照）。セクション 1.2 で説明した、直接に n 項表記する方法では $n + 1$ 個の名前しか付ける必要がないのに対し、この場合は $2n - 1$ 個を要する。例えば、図 1 の 4 項リレーション「供給」においては、n-項表記法にて 5 個の名前を要しているが、入れ子になった 2 項表記法を用いると

P(供給業者, *Q*(部品, *R*(プロジェクト, 数量)))

という形で表現され、従って、7 個の名前を用いる。

この種の表現法はさらに、非対称性という欠点も持つ。この場合の非対称性は、対称的な探索を禁じるものではないが、問い合わせの基本のいくつかは、ユーザにとって表現するのが大変厄介なものとなることが確実である（例えば、ある所与のプロジェクトに関係する部品とその数量を、*Q* と *R* を経由して問い合わせることを考えて見よ）。

1.6. 表現可能、名前付き、及び、保管されたリレーション

和訳省略　名前付きセットの定義。
名前付きリレーションは、今日のテーブルとビューに該当する。名前付きセットは、名前付きリレーション全ての集合。

⑦ 訳注：有効経路(directed path)とは、要するに矢印のついた接続線。通常、ネットワーク型モデルでの接続線には向きがあり、一方向にしか辿れない。そのため 2 項関係を両側から探索するのに、2 つの経路が必要となる。

⑧ 訳注：例えば 4 次のリレーションの場合、経路の中で最初に辿る引数（ドメイン）の選択肢が 4 つ、次に辿る引数の選択肢が残り 3 つ、…というように、辿り方のパターン（経路）の数は 4 つの要素の順列となるので、その値は 4 の階乗である。

users can identify by means of a simple name (or identifier). A relation R acquires membership in the named set when a suitably authorized user declares R; it loses membership when a suitably authorized user cancels the declaration of R.

The expressible set is the total collection of relations that can be designated by expressions in the data language. Such expressions are constructed from simple names of relations in the named set; names of generations, roles and domains; logical connectives; the quantifiers of the predicate calculus; [6] and certain constant relation symbols such as =, >. The named set is a subset of the expressible set--usually a very small subset.

和訳省略　表現可能セットの定義。
表現可能なリレーションとは、名前付きリレーションをもとに、データ副言語で表現可能なあらゆるリレーション（問い合わせ結果）。
名前付きリレーションは表現可能なリレーションでもある。
表現可能なリレーションすべての集合を、「表現可能セット」と呼ぶ。

[6] Because each relation in a practical data bank is a finite set at every instant of time, the existential and universal quantifiers can be expressed in terms of a function that counts the number of elements in any finite set.

Since some relations in the named set may be time-independent combinations of others in that set, it is useful to consider associating with the named set a collection of statements that define these time-independent constraints. We shall postpone further discussion of this until we have introduced several operations on relations (see Section 2).

和訳省略　名前付きリレーションの間に常に成立する制約を記述して登録しておくというアイデアの説明。セクション 2 でさらに検討する旨。

One of the major problems confronting the designer of a data system which is to support a relational model for its users is that of determining the class of stored representations to be supported. Ideally, the variety of permitted

ユーザ向けにリレーショナルモデルをサポートするデータシステムの設計者が直面する主要な問題のひとつは、サポートされるべき保管形式のクラス⑨を決定することである⑩。理想を言えば、許容されるデータ表現形式の種類は、[データシス

⑨ クラスとは集合の集まりを言う。ひとつの保管形式自体が集合となり、その集まりがクラスとなるという意味でクラスという用語を用いていると思われる。

⑩ 本論文に先立つ 1969 年の論文では、保管されたリレーションは名前付きリレーションの一種として位置付けられている。本論文では、ユーザ向けのデータモデルと保管用のデータ構造を切り離そうとする傾向が感じられる。

data representations should be just adequate to cover the spectrum of performance requirements of the total collection of installations. Too great a variety leads to unnecessary overhead in storage and continual reinterpretation of descriptions for the structures currently in effect.

For any selected class of stored representations the data system must provide a means of translating user requests expressed in the data language of the relational model into corresponding --and efficient-- actions on the current stored representation. For a high level data language this presents a challenging design problem. Nevertheless, it is a problem which must be solved--as more users obtain concurrent access to a large data bank, responsibility for providing efficient response and throughput shifts from the individual user to the data system.

テムの]すべての設置例での性能要件の振幅をカバーするのに丁度適合するだけの多様性を備えているべきだ。多様性が大きすぎると、不要な間接処理負荷がストレージにかかり、かつ、現在適用されている構造に合わせて、記述[データ]を絶え間なく再翻訳することになってしまう。

選択された保管形式のクラスいずれについても、データシステムは、リレーショナルモデルのデータ言語で表現されたユーザの要求を、現在の保管形式の上での、それに対応する―そして効率的な―アクションに翻訳する手段を備えなければならない。高水準のデータ言語については、これは挑戦的な設計問題となる。とはいえ、これは解決されねばならない問題である－大規模データバンクに同時アクセスするユーザの数が増えるにつれ、効率的なレスポンスとスループットを提供する責任は、個々のユーザの側からデータシステムの側へと移るのだから。

2. Redundancy and Consistency

2.1. OPERATIONS ON RELATIONS

2. 冗長性と一貫性

2.1. リレーション上の演算

訳者注解　リレーション演算の説明が、冗長性と一貫性のセクションの冒頭に置かれている点に留意。コッド博士は、リレーション演算を、冗長性の定義と一貫性制御のための基礎ツールと位置付けていたと思われる。一方、問い合わせに関しては、セクション１で、述語論理をもとにした言語が提案済みである。リレーション演算と述語論理は同等の能力を持つことが、後に証明され、現代では SQL が両方の役割を担っているが、RDBMS による冗長性定義と一貫性制御のサポートは不十分な感じもする。

Since relations are sets, all of the usual set operations are applicable to them. Nevertheless, the result may not be a relation; for example, the union of a binary relation and a ternary relation is not a relation.

リレーションは集合なので、通常の集合演算のすべてを適用することができる。とはいえ、その結果がリレーションになるとは限らない；例えば、2 項リレーションと 3 項リレーションの和集合は、リレーションではない。

The operations discussed below are specifically for relations. These operations are introduced because of their key role in deriving relations from other relations. Their principal application is in noninferential information systems-systems which do not provide logical inference services--although their applicability is not necessarily destroyed when such services are added.

以下で論じられる演算は、リレーション専用のものである。これらの演算を導入するのは、リレーションを他のリレーションから導出する上で鍵となる役割をそれらが担うからである。これらの主な適用対象は、非推論的な情報システム—論理推論サービスを提供しないシステム—であるが、そういったサービスが付加されたからといって、これらの演算の適用可能性が台無しになると決まっているわけではない。

Most users would not be directly concerned with these operations. Information systems designers and people concerned with data bank control should, however, be thoroughly familiar with them.

たいていのユーザは、これらの演算にじかに関わることはないだろう。しかし、情報システム設計者、およびデータバンクの制御に関心を持つ人々は、これらに精通していなければならない。

2.1.1. *Permutation*. A binary relation has an array representation with two columns. Interchanging these columns yields the converse relation. More generally, if a permutation is applied to the columns of an *n*-ary relation, the resulting relation is said to be a *permutation* of

2.1.1.*列順交換*.　2 項リレーションは、2 つの列を持つ配列で表現できる。これらの列の順序を交換することによって、逆リレーションが得られる。より一般化して言えば、n 項リレーションの列に 1 回の列順交換を適用した場合、結果として得られるリレーションは、もとのリレーションの*列*

the given relation. There are, for example, 4! =24 permutations of the relation *supply* in Figure 1, if we include the identity permutation which leaves the ordering of columns unchanged.

Since the user's relational model consists of a collection of relationships (domain-unordered relations), permutation is not relevant to such a model considered in isolation. It is, however, relevant to the consideration of stored representations of the model. In a system which provides symmetric exploitation of relations, the set of queries answerable by a stored relation is identical to the set answerable by any permutation of that relation. Although it is logically unnecessary to store both a relation and some permutation of it, performance considerations could make it advisable.

2.1.2. *Projection*. Suppose now we select certain columns of a relation (striking out the others) and then remove from the resulting array any duplication in the rows. The final array represents a relation which is said to be a *projection* of the given relation.

A selection operator π is used to obtain any

*順交換*であると言われる。例えば、図 1 のリレーション「供給」に対して、列の順序を変えない恒等な列順交換を含めれば、 4! =24 通りの列順交換が存在する。

ユーザ向けリレーショナルモデルは、リレーションシップ（ドメイン順序なしのリレーション）から構成されるので、そうしたモデルだけ切り離して考慮すれば、列順交換は意味が無い。しかしながら、そのモデルの保管形式を考慮する際には意味がある。対称なリレーション探索を提供するシステムにおいては、保管されたリレーションのひとつによって回答可能な問い合わせの集合は、そのリレーションのどの列順交換をとっても、その列順交換で回答可能な集合と同一である。あるリレーションとその列順交換のいくつかをともに保管することは、論理的には不要であるけれども、性能に関する配慮のゆえに、それが適切となる場合があり得る。

訳者注解　ここでコッド博士が言っているのは、保管形式においては、性能向上のためにデータの重複保持（冗長性）を許すということである。一方で、この冗長性は、ユーザ向けリレーショナルモデル上にはあらわれない。ここでいう「列順交換のいくつか」は、要するに、ユーザ向けリレーショナルモデルから隠されたインデックスファイルである。パフォーマンス対策は、ユーザ向けリレーショナルモデルではなく、その下の層で行うという発想は、後に広く適用された「非正規化」すなわち、パフォーマンス対策のためにユーザ向けリレーショナルモデルを歪めることを許容する、という発想と、ある意味、対立している。

2.1.2. *射影*.　さて、あるリレーションについて、列をいくつか選んで（他の列は抹消してしまい）、得られた配列で、重複している行があればすべて除去してしまうと想定しよう。最後に得られた配列は、もとの配列の*射影*であると言われる。

選択演算子πを用いて、所望する列順交換、射

desired permutation, projection, or combination of the two operations. Thus, if L is a list of k indices[7] $L = i_1, i_2, \ldots, i_k$ and R is an n-ary relation ($n \geq k$), then $\pi_L(R)$ is the k-ary relation whose jth column is column i_j of R ($j = 1, 2, \ldots, k$) except that duplication in resulting rows is removed. Consider the relation supply of Figure 1. A permuted projection of this relation is exhibited in Figure 4. Note that, in this particular case, the projection has fewer n-tuples than the relation from which it is derived.

π_{31}(*supply*)	(*project*	*supplier*)
	5	1
	5	2
	1	4
	7	2

FIG. 4. A permuted projection of the relation in Figure 1

[7] When dealing with relationships, we use domain names (role-qualified whenever necessary) instead of domain positions.

2.1.3. *Join.* Suppose we are given two binary relations, which have some domain in common. Under what circumstances can we combine these relations to form a ternary relation which preserves all of the information in the given relations?

影、あるいはその 2 つの演算を組み合わせた結果を得ることができる。すなわち、L が k 個の添え字[7] のリスト $L = i_1, i_2, \ldots, i_k$ とし、 R が n 項のリレーションとするならば ($n \geq k$)、そのとき、$\pi_L(R)$ は k 項のリレーションであって、その j 番目の列は、得られた行の重複が除かれることを除いて、R の列 i_j である($j = 1, 2, \ldots, k$)。図 1 のリレーション「供給」を考えよう。このリレーションの列順交換された射影のひとつが図 4 に示されている。この特定の例では、射影に含まれる n-タプルの数は、その導出元のリレーションより少ないことに留意せよ。

π_{31}(*供給*)	(*プロジェクト*	*供給業者*)
	5	1
	5	2
	1	4
	7	2

図 4. 図 1 のリレーションの列順交換後の射影のひとつ

[7] リレーションシップを扱う場合には、ドメインの位置ではなく、（必要に応じて役割で修飾した）ドメイン名を[添え字として]用いる。

訳者注解 例えば、S を 3 項リレーションとするとき、$\pi_{13}(S)$ は、S から 1 列目と 3 列目を抜き出して作った射影である。また、$\pi_{31}(S)$は、S の 1 列目と 3 列目を射影し、さらにその 2 列を列順交換した結果できたリレーションである。添え字が 2 桁の場合もあり得るので、$\pi_{31}(S)$ は、$\pi_{(3,1)}(S)$ と書くと誤解が無いと思うが、読みやすさを考慮してか、コッド博士は$\pi_{31}(S)$と書く。

2.1.3. *結合.* 2 つの 2 項リレーションがあるとせよ。両者はいずれかのドメインを共有している。いかなる状況の下なら、この 2 つのリレーションを組み合わせてひとつの 3 項リレーションを作り、それがもとの[2 つの]リレーションに含まれる全ての情報を保つようにすることができるだろうか。

訳者注解　現代のリレーショナル理論では、リレーションの情報無損失分解が問題にされるが、ここでコッド博士が定義しているのは、リレーションの情報無損失結合である。この違いは興味深い。ツリー型やネットワーク型のモデルでは、ポインタを用いて 2 つのレコードをつなぐことにより 2 項関係を表現することはできたから、3 項以上の関係を 2 項関係の結合で正しく表現できるかを問題にしたのだろう。以降でコッド博士は、2 項関係の結合が情報無損失であるにも拘らず、得られる 3 項関係がひとつに定まらない（多義性が混入する）場合があることを指摘する。つまり、3 項以上の関係には、複数の 2 項関係を結合しても表現できない情報が含まれる場合がある。これが、後に出てくる「結合のわな」の話につながる。

The example in Figure 5 shows two relations R, S, which are joinable without loss of information, while Figure 6 shows a join of R with S. A binary relation R is *joinable* with a binary relation S if there exists a ternary relation U such that $\pi_{12}(U) = R$ and $\pi_{23}(U) = S$. Any such ternary relation is called a *join* of R with S. If R, S are binary relations such that $\pi_2(R) = \pi_1(S)$, then R is joinable with S. One join that always exists in such a case is the *natural join* of R with S defined by

$$R*S = \{(a,b,c): R(a,b) \wedge S(b,c)\}$$

where $R(a, b)$ has the value true if (a, b) is a member of R and similarly for $S(b, c)$. It is immediate that

$$\pi_{12}(R*S) = R$$

and

$$\pi_{23}(R*S) = S.$$

図 5 の例は、情報損失なく結合することが可能な 2 つのリレーション R, S を示している。一方、図 6 は、R の S との結合のひとつを示している。2 項リレーション R と S について、ある 3 項リレーション U が存在して、それが、$\pi_{12}(U) = R$ かつ $\pi_{23}(U) = S$ という条件を満たす場合、R は S と*結合可能*である。このような 3 項リレーションはいずれも、R の S との*結合*と呼ばれる。こうしたケースで常に存在する結合がひとつあって R の S との*自然な結合*と呼ばれる。これは、以下のように定義される：

$$R*S = \{(a, b, c): R(a, b) \wedge S(b, c)\}$$

ここで、$R(a, b)$は、(a, b)が R の要素である場合、真となる⑪。$S(b, c)$も同様である。以上からただちに：

$$\pi_{12}(R*S) = R$$

および

$$\pi_{23}(R*S) = S.$$

とわかる。

Note that the join shown in Figure 6 is the natural join of R with S from Figure 5. Another join is shown in Figure 7.

図 6 に示される結合は、図 5 の R の S との自然な結合であることに留意せよ。それとは別の結合が図 7 に示されている。

⑪訳注：$R(a, b)$は、リレーション R を述語（＝命題関数）として表現したものである。また、論理式中の記号∧は、「かつ」すなわち AND 記号である。

R	*(supplier*	*part)*	S	*(part*	*project)*
	1	1		1	1
	2	1		1	2
	2	2		2	1

FIG. 5. Two joinable relations

R	*(供給業者*	*部品)*	S	*(部品*	*プロジェクト)*
	1	1		1	1
	2	1		1	2
	2	2		2	1

図 5. 結合可能な 2 つのリレーション

R*S	*(supplier*	*part*	*project)*
	1	1	1
	1	1	2
	2	1	1
	2	1	2
	2	2	1

FIG. 6. The natural join of R, with S (from Figure 5)

R*S	*(供給業者*	*部品*	*プロジェクト)*
	1	1	1
	1	1	2
	2	1	1
	2	1	2
	2	2	1

図 6. (図 5 の)R の S との自然な結合

U	*(supplier*	*part*	*project)*
	1	1	2
	2	1	1
	2	2	1

FIG. 7. Another join of R with S (from Figure 5)

U	*(供給業者*	*部品*	*プロジェクト)*
	1	1	2
	2	1	1
	2	2	1

図 7. (図 5 の)R の S とのもうひとつの結合

Inspection of these relations reveals an element (element 1) of the domain part (the domain on which the join is to be made) with the property that it possesses more than one relative under *R* and also under *S*. It is this element which gives rise to the plurality of joins. Such an element in the joining domain is called a *point of ambiguity* with respect to the joining of *R* with *S*.

If either $\pi_{21}(R)$ or *S* is a function, [8] no point of ambiguity can occur in joining *R* with *S*. In such a case, the natural join of *R* with *S* is the only join of *R* with *S*. Note that the reiterated qualification "of *R* with *S*" is necessary, because

これらのリレーションを注意深く観察すると、ドメイン「部品」（結合箇所となるドメイン）のひとつの要素（要素 1）が、*R* においても *S* においてもそれに関係するタプルが複数あるという特質を備えていることがわかる。結合が複数存在する事態をもたらしているのは、この要素である。結合箇所となるドメインにおけるこうした要素は、*R* の *S* との結合に際しての*多義性発生点*と呼ばれる。

$\pi_{21}(R)$ と *S* のいずれかが関数 [8] であれば、*R* の *S* との結合に際して、多義性発生点は生じえない。そのようなケースでは、*R* の *S* との*自然な結合*が、*R* の *S* との唯一の結合である。何度も繰り返される「*R* の *S* との」という修飾に留意頂きた

S might be joinable with R (as well as R with S), and this join would be an entirely separate consideration. In Figure 5, none of the relations R, $\pi_{21}(R)$, S, $\pi_{21}(S)$ is a function.

[8] A function is a binary relation, which is one-one or many-one, but not one-many.

Ambiguity in the joining of R with S can sometimes be resolved by means of other relations. Suppose we are given, or can derive from sources independent of R and S, a relation T on the domains *project* and *supplier* with the following properties:

(1) $\pi_1(T) = \pi_2(S)$,

(2) $\pi_2(T) = \pi_1(R)$,

(3) $T(j, s) \rightarrow \exists p\,(R(S, p) \wedge S(p, j))$,

(4) $R(s, p) \rightarrow \exists j\,(S(p, j) \wedge T(j, s))$

(5) $S(p, j) \rightarrow \exists s(T(j, s) \wedge R(s, p))$,

い。というのは、（R が S と結合可能なのと同様に）S は R と結合可能かもしれないが、その結合はまったく別の事柄だからだ。図 5 においては、リレーション R、$\pi_{21}(R)$、S、$\pi_{21}(S)$のいずれも、関数ではない。

[8] 「関数」は 1 対 1 あるいは多対 1 の 2 項関係である。1 対多の 2 項関係は関数ではない。

訳者注解 本論文ではまだ「関数従属性」という用語は現れないが、このパラグラフの冒頭で述べられていることは、関数従属性が含まれるリレーションは無損失分解できるということにほぼ等しい。コッド博士は、本論文執筆時点で、このことは当然と捉えていて、それよりも、この後に述べられる三方結合の問題を、従来のネットワーク型モデルなどとの対比の上で重要と考えていたのではないだろうか。実際、関数従属性による正規化は、RDB でなく、ISAM など key-value 型のファイルを用いていても、ある程度実行されていただろうから、リレーショナルモデルの価値を差別化するポイントとしては訴求しにくかっただろう。この論文に出てくるほとんどのリレーションの列は複合キーを構成していて、単なる属性項目はオマケ程度にしか出てこないことも、博士の関心の所在をしめす。リレーショナルモデルの関心の所在は、最初から、（ＥＲモデルの用語でいう）実体間の「関連」にあったのである。

他のリレーションを用いることで、R の S との結合の多義性を解消できる場合もある。ドメイン「*プロジェクト*」と「*供給業者*」上のリレーション T を、与えられるかあるいは R と S に依存しない情報源から導き出すことができ、そのリレーションは以下の特性を持つものとしよう：

(1) $\pi_1(T) = \pi_2(S)$,

(2) $\pi_2(T) = \pi_1(R)$,

(3) $T(j, s) \rightarrow \exists p\,(R(s, p) \wedge S(p, j))$,⑫

(4) $R(s, p) \rightarrow \exists j\,(S(p, j) \wedge T(j, s))$

(5) $S(p, j) \rightarrow \exists s(T(j, s) \wedge R(s, p))$,

⑫ 訳注：原文で $R(S,p)$とされている箇所は、$R(s,p)$の誤り。

then we may form a three-way join of R, S, T; that is, a ternary relation such that

$$\pi_{12}(U) = R,\ \ \pi_{23}(U) = S,\ \ \pi_{31}(U) = T.$$

この場合、R, S, Tの三方結合を作ることができ、それは、以下の条件を満たす3項リレーションとなる：

$$\pi_{12}(U) = R,\ \ \pi_{23}(U) = S,\ \ \pi_{31}(U) = T$$

訳者注解　Rの列2とSの列1、Sの列2とTの列1、さらにTの列2とRの列1を結合して、輪のように繋がった三方結合リレーションUを作るという説明。

現代の結合演算では、三方結合は常に可能だが、コッド博士の結合演算は無損失であることを要求するので、そうではなかった。説明の最後の行の3つの式は、三方結合の無損失性ーすなわち、Uにπ演算（列順交換と射影）を適用すればR, S, Tを再現できることーを示している。

条件(1)～(5)は（無損失な）三方結合が可能であることを保証する条件である。本当は、5つの条件に加えて$\pi_2(R) = \pi_1(S)$という条件があり、全部で6条件あるが、RをSと結合できることからこの条件はすでに成立していることが明らかなので、省略されている。

条件(3)～(5)は複雑に見えるが、例えば、条件(3) を日常語で言えば以下の通りである：

「ある供給業者があるプロジェクトに対して供給している(T)ならば、何らかの部品があって、そのプロジェクトはその部品を供給されており(R)、かつその供給業者はその部品を供給している(S)筈だ（その部品が何かはわからないにしても）。そうでなければ、T,R, Sは、供給業者・プロジェクト・部品の三者関係を正しく映しているとは言えない」

条件(4)・(5)は、R, S, Tの立ち位置を変えているだけで、同様のことを主張している。

なお、この箇所以降の説明で、s, p, j はそれぞれ以下の略称と考えると読みやすい：

s …　Supplier 供給業者
p …　Parts　部品
j …　proJect　プロジェクト

〔式の詳しい解説〕
条件(3)は以下のように読み下す。
「タプル(*j*, *s*)が *T*に含まれるならば、タプル(*s*, *p*)が *R* に含まれ、かつ、タプル (*p*, *j*)が *S*に含まれるような *p* が少なくともひとつ存在すること」
下線部の条件が成り立たない場合、*R* を *S* と結合できたとしても、その結果得られたリレーションには、左端（第 1 項）が *s* であり、右端（第 3 項）が *j* であるタプルを含み得ない。そのため、*T*に含まれるタプル(*j*, *s*)には、両端ともに結合できる相手タプルが無いことになる。この場合、現代的な、無損失を要件としない結合は可能だが、その際、タプル(*j*, *s*)の情報が失われてしまう。つまり損失が生じる。一方、コッド博士の結合は無損失を要件とするので、そもそも結合できないことになるのである。

Such a join will be called a cyclic 3-join to distinguish it from a linear 3-join which would be a quaternary relation *V* such that

$$\pi_{12}(V) = R,\quad \pi_{23}(V) = S,\quad \pi_{34}(V) = T.$$

このような結合は環状 3 者結合と呼ぶことにして、以下の条件を満たす 4 項リレーションである線形 3 者結合と区別しよう：

$$\pi_{12}(V) = R,\quad \pi_{23}(V) = S,\quad \pi_{34}(V) = T.$$

訳者注解　線形 3 者結合は、環状 3 者結合の、尻尾と頭をつなぐ前の形である。

While it is possible for more than one cyclic 3-join to exist (see Figures 8, 9, for an example), the circumstances under which this can occur entail much more severe constraints than those for a plurality of 2-joins.

ふたつ以上の環状 3 者結合が存在することはあり得るが（例えば図 8,9 を見よ）、それが起き得る状況は、2 者結合が複数存在する状況よりはるかに厳しい制約を課されたものになる。

R	*(s*	*p)*	S	*(p*	*j)*	T	*(j*	*s)*
	1	a		a	d		d	1
	2	a		a	e		d	2
	2	b		b	d		e	2
				b	e		e	2

FIG. 8. Binary relations with a plurality of cyclic 3 joins

R	*(s*	*p)*	S	*(p*	*j)*	T	*(j*	*s)*
	1	a		a	d		d	1
	2	a		a	e		d	2
	2	b		b	d		e	2
				b	e			

図 8.環状 3 者結合を複数持つ 2 項リレーション[13]

[13] 原文では、T のタプル(e,2)が重複している。明らかな誤りなので訳文では最後のタプルを削除した。

U	(s	p	j)
	1	a	d
	2	a	e
	2	b	d
	2	b	e

U'	(s	p	j)
	1	a	d
	2	a	d
	2	a	e
	2	b	d
	2	b	e

FIG. 9. Two cyclic 3-joins of the relations in Figure 8

U	(s	p	j)
	1	a	d
	2	a	e
	2	b	d
	2	b	e

U'	(s	P	j)
	1	a	d
	2	a	d
	2	a	e
	2	b	d
	2	b	e

図 9. 図 8 のリレーションによる 2 つの環状 3 者結合

To be specific, the relations R, S, T must possess points of ambiguity with respect to joining R with S (say point x), S with T (say y), and T with R (say z), and, furthermore, y must be a relative of x under S, z a relative of y under T, and x a relative of z under R. Note that in Figure 8 the points $x = a$; $y = d$; $z = 2$ have this property.

The natural linear 3-join of three binary relations R, S, T is given by

$$R^*S^*T = \{(a,b,c,d): R(a,b) \wedge S(b,c) \wedge T(c,d)\}$$

where parentheses are not needed on the left-hand side because the natural 2-join (*) is associative. To obtain the cyclic counterpart, we introduce the operator γ which produces a relation of degree n-1 from a relation of degree n by tying its ends together. Thus, if R is an n-ary relation ($n \geq 2$), the *tie* of R is defined by the equation

$$\gamma(R) = \{(a_1, a_2, \ldots, a_{n-1}): R(a_1, a_2, \ldots, a_{n-1}, a_n) \wedge a_1 = a_n\}.$$

We may now represent the natural cyclic 3-join of R, S, T by the expression

$$\gamma(R^*S^*T).$$

具体的に言えば、リレーション R, S, T は、多義性発生点を、R の S との結合に関して有し（x 点とする）、S の T との結合に関しても（y とする）、T の R との結合に関しても（z とする）有する必要があり、のみならず、y は S のもとで x に関係づけられ、z は T のもとで y に、x は R のもとで z に関係づけられていなければならない。図 8 においては、$x = a$; $y = d$; $z = 2$ がこの特質を備えていることに留意せよ。

3 つの 2 項リレーション R, S, T の、自然な線形 3 者結合は次の式で与えられる：

$$R^*S^*T = \{(a,b,c,d): R(a,b) \wedge S(b,c) \wedge T(c,d)\}$$

ここで、左辺に括弧付けは必要ない。自然な 2 者結合（*）では、結合則が成り立つからである。これの環状版の相方を得るために、演算子 γ（ガンマ）を導入する。これは、n 次のリレーションの両端を結んで n-1 次のリレーションを作り出す。すなわち、R が n 項リレーション($n \geq 2$)であるとして、R の*結環*[14]を次の等式で定義する：

$$\gamma(R) = \{(a_1, a_2, \ldots, a_{n-1}): R(a_1, a_2, \ldots, a_{n-1}, a_n) \wedge a_1 = a_n\}$$

さてこれで、自然な環状 3 者結合を次の式で表すことができる：

$$\gamma(R^*S^*T)$$

訳者注解　結環の過程で、先頭と最後の列の値が異なるタプルがリレーションから除去される。すなわち、結環は損失的な演算である（結環は組合

[14] 元の英語は ”tie”。リレーションの尻尾と頭を結んで環を作る演算なので「結環」と訳した。

せを絞り込むための演算なので、損失的であってしかるべきである）。
コッド博士流の無損失な結合の組合せでは損失的な演算は実現できないので、別の演算として定義したのである。

Extension of the notions of linear and cyclic 3-join and their natural counterparts to the joining of n binary relations (where $n > 3$) is obvious. A few words may be appropriate, however, regarding the joining of relations which are not necessarily binary. Consider the case of two relations R (degree r), S (degree s) which are to be joined on p of their domains ($p < r$, $p < s$). For simplicity, suppose these p domains are the last p of the r domains of R, and the first p of the s domains of S. If this were not so, we could always apply appropriate permutations to make it so. Now, take the Cartesian product of the first $r - p$ domains of R, and call this new domain A. Take the Cartesian product of the last p domains of R, and call this B. Take the Cartesian product of the last $s - p$ domains of S and call this C.

We can treat R as if it were a binary relation on the domains A, B. Similarly, we can treat S as if it were a binary relation on the domains B, C. The notions of linear and cyclic 3-join are now directly applicable. A similar approach can be taken with the linear and cyclic n-joins of n relations of assorted degrees.

線形および環状の 3 者結合とその自然版の概念の n 個の 2 項リレーション(ただし $n > 3$)への拡張はおのずから明らかである。しかし、必ずしも 2 項でない複数のリレーションの結合に関しては、二、三の言葉を加えるのが適切かもしれない。2 つのリレーション R(次数 r), S(次数 s)のケースを考えよう。p 個のドメインを結合箇所として、これら二つを結合するものとする($p < r$, $p < s$)。話を単純にするために、これら p 個のドメインは、R のドメインの最後の p 個であり、かつ、S のドメインの最初の p 個であるとする。もしそうでなければ、列順変換を適切に適用してそうなるようにすることが、常に可能である。さて、R の最初の $r - p$ 個のドメインのデカルト積をとり、この新しいドメインを A と呼ぼう。R の最後の p 個のドメインのデカルト積をとり、これを B と呼ぼう。S の最後の $s - p$ 個のドメインのデカルト積をとり、これを C と呼ぼう。

私たちは、R を、ドメイン A,B 上の 2 項リレーションであるかのように取り扱うことができ、同様に、S を、ドメイン B,C 上の 2 項リレーションであるかのように取り扱うことができる。これで、線形および環状の 3 者結合の諸概念がそのまま適用可能となった。様々な次数を持つ n 個のリレーションの線形および環状の n-結合に対しても同様のアプローチを採ることができる。

2.1.4. *Composition*. The reader is probably familiar with the notion of composition applied to functions. We shall discuss a generalization of that concept and apply it first to binary relations. Our definitions of composition and composability are based very directly on the definitions of join and joinability given above.

2.1.4. *合成*. 読者はたぶん、関数に適用される合成の概念に親しんでいるだろう。私たちはこの概念を一般化したものについて論じ、手始めに 2 項リレーションに適用する。合成と合成可能性の、私たちの定義は、すでに述べられた結合と結合可能性の定義に直接立脚するものである。

訳者注解　ここでいう合成とは、X と Y の関係および Y と Z の関係をもとに、X と Z の関係を導くことを意味している。合成は、結合と射影を続けて適用することで実現できるので不可欠な演算ではない。現代の視点からすると、たいして便利でもない。コッド博士が合成をあえて詳しく説明するのは、これが「結合のわな」の話の直接の前置きとなるからであろう。ネットワーク型モデルでレコードタイプ X から Y へ、Y から Z へと辿る操作は、論理的にはリレーションの（自然な）合成に相当する。そのため、合成に関する問題は、そのまま、ネットワーク型モデルの問題となる。コッド博士の功績のひとつは、このように、従来は、操作と捉えられていた事柄を、数学的な演算として定義し直し、批判し易くしたことにあると思う。

Suppose we are given two relations *R*, *S*. *T* is a *composition* of *R* with *S* if there exists a join *U* of *R* with *S* such that $T = \pi_{13}(U)$. Thus, two relations are composable if and only if they are joinable. However, the existence of more than one join of *R* with *S* does not imply the existence of more than one composition of *R* with *S*.

2 つのリレーション *R*, *S* を与えられたとせよ。*R* の *S* との結合 *U* が存在し、$T = \pi_{13}(U)$であるとき、*T* は *R* の *S* との*合成*である。従って、2 つのリレーションは、結合可能な場合、かつその場合に限り合成可能である。とはいえ、*R* の *S* との結合が複数存在することは、*R* の *S* との合成が複数存在することを含意しない。

Corresponding to the natural join of *R* with *S* is the *natural composition*[9] of *R* with *S* defined by

R の *S* との自然な結合に対応して、*R* の *S* との*自然な合成*[9]があり、以下のように定義される：

$R \cdot S = \pi_{13}(R^*S)$.

Taking the relations *R*, *S* from Figure 5, their natural composition is exhibited in Figure 10 and another composition is exhibited in Figure 11 (derived from the join exhibited in Figure 7).

図 5 のリレーション *R*, *S* を採れば、その自然な合成は図 10 に示され、それとは異なる合成が図 11 に示される（これは図 7 の結合をもとに導出される）。

R*S	*(project*	*supplier)*
	1	1
	1	2
	2	1
	2	2

FIG. 10. The natural composition of R with S (from Figure 5)

R*S	*(供給業者*	*プロジェクト)*
	1	1
	1	2
	2	1
	2	2

図 10.　（図 5 の）R の S との自然な合成[15]

[15] 訳注：原文では、第 1 項が project、第 2 項が supplier であるが、図 5 と照らし合わせると逆である。内容は、たまたま、いずれでも正しいものとなっている。図 11 も同じ。

T	*(project*	*supplier)*
	1	2
	2	1

FIG. 11. Another composition of R with S (from Figure 5)

T	*(供給業者*	*プロジェクト)*
	1	2
	2	1

図 11. （図 5 の）R の S とのもうひとつの合成

[9] Other writers tend to ignore compositions other than the natural one, and accordingly refer to this particular composition as the composition--see, for example, Kelley's "General Topology."

When two or more joins exist, the number of distinct compositions may be as few as one or as many as the number of distinct joins. Figure 12 shows an example of two relations which have several joins but only one composition. Note that the ambiguity of point *c* is lost in composing *R* with *S*, because of unambiguous associations made via the points *a*, *b*, *d*, *e*.

[9] 他の著述者は、自然な合成以外の合成を無視し、結果として、この特定の合成（訳注：自然な合成）を単に合成と呼ぶきらいがある。一例えば、Kelley の “General Topology ”を見よ。

2 個以上の結合が存在するとき、異なる合成の数は、最も少なくて 1、最も多い場合には、異なる結合の数である。図 12 は、結合を数個持つが合成は 1 個しか持たない 2 つのリレーションの例を示している。*R* を *S* と結合する際に、点 *c* における多義性が、点 *a*, *b*, *d*, *e* により作られる多義性のない関連づけによって失われることに留意せよ。

R	*(supplier*	*part)*	S	*(part*	*project)*
	1	a		a	g
	1	b		b	f
	1	c		c	f
	2	c		c	g
	2	d		d	g
	2	e		e	f

FIG. 12. Many joins, only one composition

R	*(供給業者*	*部品)*	S	*(部品*	*プロジェクト)*
	1	a		a	g
	1	b		b	f
	1	c		c	f
	2	c		c	g
	2	d		d	g
	2	e		e	f

図 12. 多数の結合、唯一の合成

Extension of composition to pairs of relations which are not necessarily binary (and which may be of different degrees) follows the same pattern as extension of pairwise joining to such relations.

必ずしも 2 項でない（そして異なる次数を持つこともある）リレーションの対を扱えるよう合成を拡張するには、対結合（訳注：2 者結合のこと）をそのようなリレーション対に拡張したときと同じパターンに従う。

A lack of understanding of relational composition has led several systems designers into what may be called the connection trap. This trap may be described in terms of the following example. Suppose each supplier description is linked by pointers to the descriptions of each part supplied by that

リレーションの合成に関する理解が欠けているために、数人の設計者が、*結合のわな*とでも呼べるものに、これまで陥ってきた。次の例を用いればこのわなをうまく説明することができるかもしれない。供給業者の記述（訳注：レコードのこと）は、おのおの、ポインタによって、その供給者が供給している各部品の記述に紐づけられてい

supplier, and each part description is similarly linked to the descriptions of each project which uses that part. A conclusion is now drawn which is, in general, erroneous: namely that, if all possible paths are followed from a given supplier via the parts he supplies to the projects using those parts, one will obtain a valid set of all projects supplied by that supplier. Such a conclusion is correct only in the very special case that the target relation between projects and suppliers is, in fact, the natural composition of the other two relations--and we must normally add the phrase "for all time," because this is usually implied in claims concerning path-following techniques.

る。そして、各部品の記述は、同様に、おのおの、その部品を使用している各プロジェクトの記述に紐づけられている。ここで、ひとつの結論が導かれるのだが、それが、一般論として、間違っているのだ。その結論とは、任意の供給業者を起点として、その供給業者が供給する部品、次にその部品を使用するプロジェクト、という具合に辿れる経路をすべて辿れば、その供給業者の供給先であるプロジェクトすべてを含む集合が正しく得られる、というものだ。このような結論が正しいのは、求めようとしているところの、プロジェクトと供給業者のリレーションが、実際に、他の 2 つのリレーションの自然な合成であるという、極めて特殊なケースに限られる。－しかも、通常は、「常に」というフレーズをそれに付け加えなければならない。というのは、経路追跡テクニックにかかわる主張には、普通、それが含意されているからだ。

訳者注解　有名な「結合のわな」の話である。通常、この話は「2 項リレーション 2 つの自然な結合で作られた 3 項リレーションには意味のないタプルが含まれる可能性があるので注意が必要」という語り口で語られる。しかしそう読むと、リレーショナルモデルのメリットにつながらない。先にも述べたように、コッド博士はこれを、ツリー型・ネットワーク型モデルへの批判として書いている。すなわち、これらのモデルでは、関係はポインタで表現されているが、ポインタは 2 項関係しか表現できない一方、現実の関係には 2 項関係の結合で表現できない 3 項以上の関係があるということを主張しているのだと思う。なお、例えばネットワーク型でも 3 者以上の関係を正しく表現することはできる。ただし、そのためには、レコードを結ぶポインタで関係を表すのではなく、関係そのものを表すレコードを設けなければならない（それを 3 者とポインタでつなぐ）。すなわち、ネットワーク型では、2 項関係はポインタで、3 項以上の関係はレコードで表現することになって一貫性がない。

2.1.5. *Restriction*. A subset of a relation is a relation. One way in which a relation S may act on a relation R to generate a subset of R is through the operation *restriction* of R by S. This operation is a generalization of the restriction of a function to a subset of its domain, and is defined as follows.

Let L, M be equal-length lists of indices such that $L = i_1, i_2, \cdots, i_k$, $M = j_1, j_2, \cdots, j_k$ where $k \leqq$ degree of R and $k \leqq$ degree of S. Then the L, M restriction of R by S denoted $R_L|_M S$ is the maximal subset R' of R such that

$$\pi_L(R') = \pi_M(S).$$

The operation is defined only if equality is applicable between elements of $\pi_{i_h}(R)$ on the one hand and $\pi_{j_h}(S)$ on the other for all $h = 1, 2, \ldots, k$.

The three relations R, S, R' of Figure 13 satisfy the equation $R' = R_{(2,3)}|_{(1,2)} S$.

R	(s	p	j)	S	(p	j)	R'	(s	p	j)
	1	a	A		a	A		1	a	A
	2	a	A		c	B		2	a	A
	2	a	B		b	B		2	b	B
	2	b	A							
	2	b	B							

FIG. 13. Example of restriction

We are now in a position to consider various applications of these operations on relations.

2.2. REDUNDANCY

Redundancy in the named set of relations must be distinguished from redundancy in the stored set of representations. We are primarily concerned here with the former. To begin with, we need a precise notion of derivability for relations.

Suppose θ is a collection of operations on relations and each operation has the property

2.1.5. *制限*. リレーションの部分集合はリレーションである。リレーション S をリレーション R に作用させて R の部分集合を作る方法のひとつに、S による R の*制限*演算がある。この演算は、関数を定義域の部分集合に制限することの一般化であって、次のように定義される。

L, M を長さの等しい添え字のリストとしよう。$L = i_1, i_2, \cdots, i_k$, $M = j_1, j_2, \cdots, j_k$ であって、$k \leqq$ R の次数、かつ、$k \leqq$ S の次数とする。R の S による L, M 制限は、$R_L|_M S$ と書かれ、次の条件を満たす、R の最大の部分集合 R' である

$$\pi_L(R') = \pi_M(S).$$

この演算は、すべての $h = 1, 2, \ldots, k$ について、$\pi_{i_h}(R)$ の要素と $\pi_{i_h}(R)$ の要素の間に等価性が適用できる場合にのみ定義される。

図 13 の 3 つのリレーション R, S, R' は、等式 $R' = R_{(2,3)}|_{(1,2)} S$ を満たす。

R	(s	p	j)	S	(p	j)	R'	(s	p	j)
	1	a	A		a	A		1	a	A
	2	a	A		c	B		2	a	A
	2	a	B		b	B		2	b	B
	2	b	A							
	2	b	B							

図 13. 制限の例

さてこれで、私たちは、これらの演算のさまざま適用方法について考えるべき位置にいる。

2.2. 冗長性

名前付きセットにおける冗長性は、保管されたデータ表現のセットにおける冗長性と区別されねばならない。私たちがここで関心を持つのは主として前者についてである。手始めに、私たちはリレーションの導出可能性の正確な定義を必要としている。

リレーションに対する演算の集合ひとつをとって θ としよう。また、各演算はその被演算子を

that from its operands it yields a unique relation (thus natural join is eligible, but join is not). A relation R is θ-*derivable* from a set S of relations if there exists a sequence of operations from the collection θ which, for all time, yields R from members of S. The phrase "for all time" is present, because we are dealing with time-varying relations, and our interest is in derivability which holds over a significant period of time. For the named set of relationships in noninferential systems, it appears that an adequate collection θ_1 contains the following operations: projection, natural join, tie, and restriction. Permutation is irrelevant and natural composition need not be included, because it is obtainable by taking a natural join and then a projection. For the stored set of representations, an adequate collection θ_2 of operations would include permutation and additional operations concerned with subsetting and merging relations, and ordering and connecting their elements.

2.2.1. *Strong Redundancy*. A set of relations is *strongly redundant* if it contains at least one relation that possesses a projection which is derivable from other projections of relations in the set. The following two examples are intended to explain why strong redundancy is defined this way, and to demonstrate its practical use. In the first example the collection of relations consists of just the following relation:

employee (serial #, name, manager#,managername)

with serial# as the primary key and manager# as a foreign key. Let us denote the active domain by Δ_t and suppose that

$$\Delta_t(\text{manager\#}) \subset \Delta_t(\text{serial\#})$$

and

$$\Delta_t(\text{managername}) \subset \Delta_t(\text{name})$$

for all time t. In this case the redundancy is

もとに、一意なリレーションを生じるものとしよう（かくして、自然な結合は適格だが、結合は非適格となる）。リレーションの集合 S のメンバいくつかをもとにして常にリレーション R を生じるような、集合 θ に属する演算子の並びが存在するならば、R は S から θ-*導出可能*である、という。「常に」というフレーズが含まれているのは、私たちは時の経過に伴い変化するリレーションを相手にしており、非常に長い時間にわたって有効であり続けるような導出可能性が、私たちの興味の対象であるからだ。非推論的システムにおける名前付きリレーションシップの集合については、適切な集まり θ_1 は次の演算、すなわち射影、自然な結合、結環、および制限を含むように思える。列順交換は無意味である。自然な合成は含める必要が無い、というのは、自然な結合の射影をとれば得られるからである。保管されたデータ表現の集合について言えば、適切な集まり θ_2 は、列順変換、リレーションの部分集合生成と合併、およびリレーションの要素の順序付けと関連付けに関する追加的な演算を含むだろう。

2.2.1. *強い冗長性*. リレーションの集合は、それに含まれるリレーションのうち少なくとも 1 つが、その集合に含まれるリレーションの他のいくつかの射影から導出可能な射影を持つ場合、*強く冗長*である。次の 2 つの例は、強い冗長性がこのように定義される理由を説明し、その実際的用途を示すことを意図している。最初の例では、リレーションの集まりは、次のリレーションのみで構成されている：

従業員（連番, 氏名, 上司連番, 上司氏名）

ここで、連番が（ザ付きの）プライマリキーであり、上司連番は外部キーである。アクティブドメインを Δ_t と表記し、あらゆる時点 t において

$$\Delta_t(\text{上司連番}) \subset \Delta_t(\text{連番})$$

かつ、

$$\Delta_t(\text{上司氏名}) \subset \Delta_t(\text{氏名})$$

が成り立つと想定しよう。このケースでは、冗長

obvious: the domain *managername* is unnecessary. To see that it is a strong redundancy as defined above, we observe that

$\pi_{34}(employee) = \pi_{12}(employee)_1 |_1 \pi_3(employee)$.

In the second example the collection of relations includes a relation S describing suppliers with primary key *s#*, a relation D describing departments with primary key *d#*, a relation J describing projects with primary key *j#*, and the following relations:

$P(s\#, d\#, \ldots)$, $Q(s\#, j\#, \ldots)$, $R(d\#, j\#, \ldots)$,

where in each case ... denotes domains other than *s#*, *d#*, *j#*. Let us suppose the following condition C is known to hold independent of time: supplier s supplies department d (relation P) if and only if supplier s supplies some project j (relation Q) to which d is assigned (relation R). Then, we can write the equation

$\pi_{12}(P) = \pi_{12}(Q) \cdot \pi_{12}(R)$

and thereby exhibit a strong redundancy.

性は明らかである：ドメイン*上司氏名*は不要だ。これが、先に定義された強い冗長性であることを理解するには、

$\pi_{34}(\text{従業員}) = \pi_{12}(\text{従業員})_1 |_1 \pi_3(\text{従業員})$.

であることに注意すればよい。

訳者注解　この例で説明されているのは、実質的には関数従属性に基づく正規化である（ただし、*上司連番*と*上司氏名*の間の関数従属性が明示されていないので論理に少し飛躍がある）。

二番目の例では、リレーションの集合は、供給業者を記述し、*s#*をプライマリキーとするリレーション S、部門を記述し、*d#*をプライマリキーとするリレーション D、プロジェクトを記述し、*j#*をプライマリキーとするリレーション J、および次のリレーションを含んでいる：

$P(s\#, d\#, \ldots)$, $Q(s\#, j\#, \ldots)$, $R(d\#, j\#, \ldots)$,

なおそれぞれのリレーションで、... は、*s#*, *d#*, *j#* 以外のドメインを示している。次の条件 C が、時点を問わず成り立つことが知られているものとしよう：供給業者 s が部門を d に供給する（リレーション P）のは、供給業者 s がいずれかのプロジェクト j に供給し（リレーション Q）、そのプロジェクト j に d が参加している（リレーション R）場合であり、またその場合に限る。このとき、下記の等式：

$\pi_{12}(P) = \pi_{12}(Q) \cdot \pi_{12}(R)$

を書くことができ、この式によって強い冗長性が示される。

訳者注解　これは、結合のわなを心配せずに、リレーション 2 つ（$\pi_{12}(Q)$ と $\pi_{12}(R)$）の自然な合成を信用することができる特殊ケースである（条件 C がそれを可能にしている）。

ここでの例のように、あるリレーション X が複数のリレーションの自然な結合であるとき、現代では、X には*結合従属性(JD)*があるといい、結合従属性のないリレーションを第 5 正規形と呼ぶ。第 5 正規形は、第 1～第 3 正規形でもあるから、この時点でコッド博士は第 5 正規形まで見通していて、

それを部分的に具体化したのが、後の関数従属性と正規形の定義となったとも言える。
ところでこの例の場合、$\pi_{12}(P)$は確かに Q と R から導出できるが、Pの全体はそうではない（従属項があるので）。だから、実際には Pを消去することはできず、この冗長性も排除できない。こうした問題をコッド博士はどう考えたのだろう。

An important reason for the existence of strong redundancies in the named set of relationships is user convenience. A particular case of this is the retention of semi-obsolete relationships in the named set so that old programs that refer to them by name can continue to run correctly. Knowledge of the existence of strong redundancies in the named set enables a system or data base administrator greater freedom in the selection of stored representations to cope more efficiently with current traffic. If the strong redundancies in the named set are directly reflected in strong redundancies in the stored set (or if other strong redundancies are introduced into the stored set), then, generally speaking, extra storage space and update time are consumed with a potential drop in query time for some queries and in load on the central processing units.

リレーションシップの名前付きセットに強い冗長性が存在する重要な理由のひとつは、ユーザの利便性である。とりたてて例を挙げれば、半分陳腐化したリレーションシップを名前付きセットに保持し続け、それを名前で参照する古いプログラムが正しく動作し続けられるようにするケースがある。システムあるいはデータベース管理者は、名前付きセット中に強い冗長性が存在することを知っておけば、現在のトラフィックにさらに効率的に対処するために保管形式を選択する際に、より大きな自由度を手にすることができる。名前付きセット中の強い冗長性が、保管セット中の強い冗長性に直接反映される（か、他の強い冗長性が保管セットに持ち込まれる）ならば、一般的に言って、余分な記憶スペースと更新時間が消費されるとともに、いくつかの問い合わせについての問い合わせ時間が短縮され中央演算処理装置（CPU）の負荷が縮小される可能性がある。

訳者注解　歯切れの悪いパラグラフである。ここでは、陳腐化しかけているリレーションうんぬんといった現実対応的な話ではなく、純理論的に言って、やろうと思えば名前付きセットからすべての冗長性を除去できるのか、あるいは除去できないとすればどんな場合か突き詰めて欲しかった（例えば正規化にしても、現実にはできない場合がある。そのことと、正規化理論を確立することの重要性は別物である）。直前パラグラフの２番目の冗長性は、純理論的に考えても名前付きセットから除去できないように見えるだけに、なおさらである。
なお、歯切れ悪さの要因のひとつとして、名前付きセット中で、C.J.デイトの言う「基底リレーショ

2.2.2. *Weak Redundancy*. A second type of redundancy may exist. In contrast to strong redundancy it is not characterized by an equation. A collection of relations is weakly redundant if it contains a relation that has a projection which is not derivable from other members but is at all times a projection of *some* join of other projections of relations in the collection.

ン」とそれ以外の「導出リレーション」が区別されていないこともあると思う。

2.2.2. *弱い冗長性*. ２番目のタイプの冗長性が存在することがある。強い冗長性と違って、そのおのおのがひとつの等式で特徴づけられるものではない。リレーションの集合に属するリレーションの射影のひとつが、他のメンバ（訳注：集合に属する他のリレーション）から導出可能ではないが、常に、その集合中のリレーションの他の射影の、*ある*結合の射影である場合、そのリレーション集合は*弱く冗長*である。

訳者注解 「*ある*結合（*some* join)」がミソである。これは自然な結合以外の結合であってかまわない。「自然な結合」に限定すれば、ここで記述されている条件が満たされる場合は強い冗長性の条件も明らかに満たされるので「弱い冗長性」を定義する意味が無い。「ひとつの等式で特徴づけられ」ないのは、結合が複数あり得るからである。そのため、弱い冗長性があったからといって、リレーション集合中に情報の重複があるわけではない。その意味では、弱い冗長性は、冗長性というより、データに関する整合性条件の一種という方が適切だろう。

We can exhibit a weak redundancy by taking the second example (cited above) for a strong redundancy, and assuming now that condition *C* does not hold at all times. The relations $\pi_{12}(P)$, $\pi_{12}(Q)$, $\pi_{12}(R)$ are complex [10] relations with the possibility of points of ambiguity occurring from time to time in the potential joining of any two. Under these circumstances, none of them is derivable from the other two. However, constraints do exist between them, since each is a projection of some cyclic join of the three of them. One of the weak redundancies can be characterized by the statement: for all time, $\pi_{12}(P)$ is *some* composition of $\pi_{12}(Q)$ with $\pi_{21}(R)$. The composition in question might be the

強い冗長性に関する（上に引いた）２番目の例を用い、今度は条件 *C* が常には成り立たないと仮定して、弱い冗長性を提示することができる。リレーション $\pi_{12}(P)$, $\pi_{12}(Q)$, $\pi_{12}(R)$は、複合リレーション [10] であって、いずれの２つを結合するにも、時おり多義発生点を生じる可能性がある。こうした条件のもとでは、いずれのリレーションも、他の２つから導出することはできない。しかし、3者の間に、制約は確かに存在する。というのは、おのおのは、3者すべてから成るある環状結合の射影だからだ。弱い冗長性のひとつは、次の文によりその特徴を言い表せる：$\pi_{12}(P)$は、常に、$\pi_{12}(Q)$の$\pi_{21}(R)$との、*ある*合成である（訳注：合成は結合の射影だから、前出の弱い冗長性の定義に即している点に注意）。その合成は、ある時点では自

natural one at some instant and a nonnatural one at another instant.

然な合成かもしれないし、別の時点ではそうでないかもしれない。

10 A binary relation is complex if neither it nor its converse is a function.

10 2項リレーションが複合リレーションであるとは、それ自身とその逆リレーションのいずれも関数でないことを言う。

訳者注解 ここで言っているのは三方結合の前提条件の(3)・(4)・(5)と同じ主張である。
*P,Q,R*が三方結合を作れるように、*P*のタプルが *Q* と *R* のタプルの組合せに対する制約として働く。言いかえれば、*P,Q,R*が、実は、リレーション集合に含まれないあるひとつの3項リレーションの影（射影）であるために、お互いに他を束縛しているのである。
三方結合の箇所で述べたのと同様、*P,Q,R*の立ち位置を変えた3つの制約条件（＝弱い冗長性）があるはずである。

Generally speaking, weak redundancies are inherent in the logical needs of the community of users. They are not removable by the system or data base administrator. If they appear at all, they appear in both the named set and the stored set of representations.

一般論を言うと、弱い冗長性は、ユーザコミュニティの論理的ニーズに属することがらである。システムやデータベース管理者が除去することはできない。弱い冗長性が出現する場合、それは、名前付きセット、保管形式のセットの双方に現れる。

訳者注解 三方結合リレーションのひとつをリレーション集合に持ち込めば、前述の三方結合にもとづく弱い冗長性は強い冗長性に変わり、除去できる（*P,Q,R*はその三方結合の射影により得られるので）。しかし、これはリレーション集合に新たな情報を持ちこむことなので、一般論として可能ではなく、ユーザの判断次第である。これが、このパラグラフの最初の二文の意図であろう。データモデルを作る立場から言えば、三者関係について、それをストレートに3項リレーションで表すのか、2つあるいは3つの2項リレーションに分けて表すのかという選択肢があるということになる。

2.3. CONSISTENCY

Whenever the named set of relations is redundant in either sense, we shall associate with that set a collection of statements which define all of the redundancies which hold independent of time between the member relations. If the information system lacks--and it most probably will--detailed semantic information about each named relation, it cannot deduce the redundancies applicable to the named set. It might, over a period of time, make attempts to induce the redundancies, but such attempts would be fallible.

Given a collection C of time-varying relations, an associated set Z of constraint statements and an instantaneous value V for C, we shall call the state (C, Z, V) *consistent* or *inconsistent* according as V does or does not satisfy Z. For example, given stored relations R, S, T together with the constraint statement "$\pi_{12}(T)$ is a composition of $\pi_{12}(R)$ with $\pi_{12}(S)$", we may check from time to time that the values stored for R, S, T satisfy this constraint. An algorithm for making this check would examine the first two columns of each of R, S, T (in whatever way they are represented in the system) and determine whether

2.3. 一貫性

訳者注解　コッド博士は、冗長性を制約ととらえて、それを満たしている状態を、一貫性がある状態と定義する。こうした定義は若干狭いように思う。例えば部品表では、完成品の親部品は存在してはならないという制約がある筈だが、これは冗長性とは関係が無い。冗長性と制約はイコールではないと思う。
一方で、データに関する制約を（手続き的にではなく）宣言的に記述するというアイデアは先進的だったし、40 年以上経った現在でも部分的にしか実現されていない。

リレーションの名前付きセットがいずれかの意味で冗長である場合、それを構成するリレーションの間に常に成り立つ冗長性すべてを定義する一群の文を、そのセットに結び付けるべきだろう。名前付きリレーションのひとつひとつについての詳細な意味的情報を欠くならば、--そして恐らくは欠くであろうが--、情報システムは、名前付きセットに当てはまる冗長性を演繹することができない。情報システムが、長い期間をかけて、冗長性を帰納的に発見すべく試みることもあり得るが、そうした試みは、当てにならないだろう。

時の経過に伴い変化するリレーションを含む集合 C、それに結び付けられた制約宣言文の集合 Z、ある時点における C の値 V を与えられた場合に、V が Z を満たす、満たさないに従い、状態(C, Z, V)には*一貫性がある*、あるいは*一貫性違反がある*と呼ぶことにしよう。例えば、保管されたリレーション R, S, T を、制約宣言文「$\pi_{12}(T)$は$\pi_{12}(R)$の$\pi_{12}(S)$との合成である。」とともに与えられたとすれば、私たちは、R, S, T に格納された値がこの制約を満たすことを、折に触れて検査することができる。検査のアルゴリズムのひとつは、R, S, T のそれぞれの最初の 2 列を調べ（その方法はそれらのシステム内の表現に応じて何でもよい）、以下が成立しているかを判断する：

(1) $\pi_1(T) = \pi_1(R)$,
(2) $\pi_2(T) = \pi_2(S)$,
(3) for every element pair (a, c) in the relation $\pi_{12}(T)$ there is an element b such that (a, b) is in $\pi_{12}(R)$ and (b, c) is in $\pi_{12}(S)$.

There are practical problems (which we shall not discuss here) in taking an instantaneous snapshot of a collection of relations, some of which may be very large and highly variable.

It is important to note that consistency as defined above is a property of the instantaneous state of a data bank, and is independent of how that state came about. Thus, in particular, there is no distinction made on the basis of whether a user generated an inconsistency due to an act of omission or an act of commission. Examination of a simple example will show the reasonableness of this (possibly unconventional) approach to consistency.

Suppose the named set C includes the relations S, J, D, P, Q, R of the example in Section 2.2 and that P, Q, R possess either the strong or weak redundancies described therein (in the particular case now under consideration, it does not matter which kind of redundancy occurs). Further, suppose that at some time t the data bank state is consistent and contains no project j such that supplier 2 supplies project j and j is assigned to department 5. Accordingly, there is no element (2, 5) in $\pi_{12}(P)$. Now, a user introduces the element (2, 5) into $\pi_{12}(P)$ by inserting some appropriate element into P. The data bank state is now inconsistent. The inconsistency could have arisen from an act of omission, if the input (2, 5) is correct, and there does exist a project j such that supplier 2 supplies j and j is assigned to department 5. In

(1) $\pi_1(T) = \pi_1(R)$,
(2) $\pi_2(T) = \pi_2(S)$,
(3) リレーション $\pi_{12}(T)$ の全てのタプル (a, c) について、(a, b) が $\pi_{12}(R)$ に含まれ、(b, c) が $\pi_{12}(S)$ に含まれるような、ある要素 b が存在する.

リレーション集合には、非常に大きく、また、変化の速いリレーションが含まれる可能性があるので、ある瞬間においてそのスナップショットを取ることには、（ここでは議論しない）実践上の問題がある。

上述のように定義された一貫性はデータバンクの瞬間的状態の属性であって、その状態が生じた経緯には関係が無いことに留意するのは重要である。特に言えば、ユーザが一貫性違反を作りだしたのが作為によるのか不作為によるのかにもとづく区別はない。単純な例をひとつ検討すれば、一貫性に対するこの（たぶん慣習にそぐわない）アプローチが理にかなっていることがわかる。

名前付きセット C に、セクション 2.2 の例にあるリレーション S, J, D, P, Q, R が含まれるとせよ。また、P, Q, R には、同セクションにて説明された強い冗長性と弱い冗長性のいずれかがあるとせよ（今検討しているこの特定のケースでは、いずれの種類の冗長性かは問題ではない）。さらに、ある時点 t において、データバンクの状態は一貫性があり、供給業者 2 がプロジェクト j に供給し、かつ、j に部門 5 が参加している、というようなプロジェクト j は無いとせよ。以上に従い、$\pi_{12}(P)$ には、要素 (2, 5) が無い。今、ユーザが、ある適当な要素を P に挿入することにより、$\pi_{12}(P)$ に要素 (2, 5) が加わる。データバンクは、いま、一貫性がない状態である。この一貫性違反は不作為により生じたのかもしれない。もし、(2,5)の入力が正しく、供給業者 2 がプロジェクト j に供給し、かつ、j に部門 5 が参加している、というようなプロジェクト j が、事実、存在するならば、そうである。こ

this case, it is very likely that the user intends in the near future to insert elements into Q and R which will have the effect of introducing $(2, j)$ into $\pi_{12}(Q)$ and $(5, j)$ in $\pi_{12}(R)$. On the other hand, the input (2, 5) might have been faulty. It could be the case that the user intended to insert some other element into P-an element whose insertion would transform a consistent state into a consistent state. The point is that the system will normally have no way of resolving this question without interrogating its environment (perhaps the user who created the inconsistency).

There are, of course, several possible ways in which a system can detect inconsistencies and respond to them. In one approach the system checks for possible inconsistency whenever an insertion, deletion, or key update occurs. Naturally, such checking will slow these operations down. If an inconsistency has been generated, details are logged internally, and if it is not remedied within some reasonable time interval, either the user or someone responsible for the security and integrity of the data is notified. Another approach is to conduct consistency checking as a batch operation once a day or less frequently. Inputs causing the inconsistencies which remain in the data bank state at checking time can be tracked down if the system maintains a journal of all state-changing transactions. This latter approach would certainly be superior if few nontransitory inconsistencies occurred.

2.4. SUMMARY

In Section 1 a relational model of data is proposed as a basis for protecting users of formatted data systems from the potentially disruptive changes in data representation caused by growth in the data bank and changes

のケースでは、恐らくユーザは近い将来、$(2, j)$ を $\pi_{12}(Q)$ に加え $(5, j)$ を $\pi_{12}(R)$に加えるような効果を持つ要素を、Q と R に挿入するつもりなのだろう。他方で、(2,5)の入力は誤りだったのかもしれない。ユーザはある別の要素を Pに挿入する意図だった—そしてその要素を挿入することで、一貫性ある状態が別の一貫性ある状態に変化するはずだった—ということもあり得る。ここでのポイントは、システムは、その環境に（たぶん一貫性違反を作り出したユーザに）問いただすことなく、この問いに答えを出すすべを、通常、持たないということである。

システムが一貫性違反を検出しそれに対応する方法は、もちろん、いくつかあり得る。ひとつのアプローチは、挿入、削除、あるいはキーの更新が起きた時にはかならず、発生し得る一貫性違反を検査するというものである。当然ながら、そうした検査は、これらの操作の速度を落とす。一貫性違反が作り出された時には、詳細が内部的にログに記録され、合理的な時間間隔をおいてもそれが是正されなかった場合には、ユーザか、セキュリティとデータ整合性に責任を持つ誰かが通知を受ける。もうひとつのアプローチは、一日に一度かもっと少ない頻度で、バッチ処理として、一貫性検査を実行するというものである。状態変更トランザクションの全てを記録したジャーナルをシステムが維持してれば、検査の時点でデータバンクに残存しており、一貫性違反の原因となっているインプットを突き止めることができる。非一過性の一貫性違反がほとんど無いのであれば、後者のアプローチの方が確実に優れているだろう。

2.4. 要約

セクション 1 では、データバンクの成長とトラフィックの変化により引き起こされる、データ表現の潜在的に破壊的な変更から、定型データシステムのユーザを防御する基盤として、データのリレーショナルモデルが提案された。時の経過に伴

in traffic. A normal form for the time-varying collection of relationships is introduced.

In Section 2 operations on relations and two types of redundancy are defined and applied to the problem of maintaining the data in a consistent state. This is bound to become a serious practical problem as more and more different types of data are integrated together into common data banks.

Many questions are raised and left unanswered. For example, only a few of the more important properties of the data sublanguage in Section 1.4 are mentioned. Neither the purely linguistic details of such a language nor the implementation problems are discussed. Nevertheless, the material presented should be adequate for experienced systems programmers to visualize several approaches. It is also hoped that this paper can contribute to greater precision in work on formatted data systems.

Acknowledgment. It was C. T. Davies of IBM Poughkeepsie who convinced the author of the need for data independence in future information systems. The author wishes to thank him and also F. P. Palermo, C. P. Wang, E. B. Altman, and M. E. Senko of the IBM San Jose Research Laboratory for helpful discussions.

RECEIVED SEPTEMBER, 1969;REVISED FEBRUARY, 1970

REFERENCES

1. CHILDS,D.L. Feasibility of a set-theoretical data structure --a general structure based on a reconstituted definition of relation. Proc. IFIP Cong., 1968, North Holland Pub. Co.,

い変化するリレーションシップ集合についての正規形が紹介された。

セクション 2 では、リレーション上の演算と 2 つのタイプの冗長性が定義され、データを一貫性ある状態に維持する問題に適用された。共通のデータバンクに統合されるそれぞれ異なるデータのタイプがますます増えるにつれ、これが現実の深刻な問題となることは明らかである。

多くの疑問が提起され、答えられないまま残された。例えば、データ副言語についてはもっと多くの重要な特性があるが、その 2、3 についてしか、セクション 1.4 では触れられなかった。そうした言語の純粋に言語的な側面や実装上の問題も論じられなかった。とはいえ、提示された素材は、経験深いシステムプログラマにとっては、いくつかのアプローチを思い描くのに十分なはずである。定型データシステムに関する研究がさらに精密なものとなることに、この論文が貢献できることも期待される。

謝辞.（以下、和訳省略）

1969 年 9 月受領. 1970 年 2 月修正.

参考文献

（和訳省略）

Amsterdam, p. 162-172.
2. LEVEIN, R. E., AND MARON, M. E. A computer system for inference execution and data retrieval. Comm. ACM 10, 11 (Nov. 1967), 715--721.
3. BACHMAN,C. W. Software for random access processing. Datamation (Apr. 1965), 36--41.
4. McGEE, W. C. Generalized file processing. In Annual Review in Automatic Programming 5, 13, Pergamon Press, New York, 1969, pp. 77-149.
5. Information Management System/360, Application Description Manual H20-0524-1. IBM Corp., White Plains, N. Y., July 1968.
6. GIS (Generalized Information System), Application Description Manual H20-0574. IBM Corp., White Plains, N. Y., 1965.
7. BLEIER, R.E. Treating hierarchical data structures in the SDC time-shared data management system (TDMS). Proc. ACM 22nd Nat. Conf., 1967, MDI Publications, Wayne, Pa., pp. 41---49.
8. IDS Reference Manual GE 625/635, GE Inform. Sys. Div., Pheonix, Ariz., CPB 1093B, Feb. 1968.
9. CHURCH,A. An Introduction to Mathematical Logic I. Princeton U. Press, Princeton, N.J., 1956.
10. FELDMAN, J. A., AND ROVNER, P.D. An Algol-based associative language. Stanford Artificial Intelligence Rep. AI-66, Aug. 1, 1968.