解　説

CODASYL 方式のデータベース・システム*

植　村　俊　亮**

0. はじめに

1970 年代に入ってからのデータベース・システム研究の高潮をもたらしたきっかけの一つは，うたがいもなく CODASYL によるデータベース用共通言語体系の開発であった．1971 年が開発の一つの区切りであり，その後5年余りの間に世界中の計算機メーカやソフトウェア会社によって実動化が積極的に行われた．CODASYL はデータベース用共通言語体系を開発したのであって，しかもそれはいわゆるデータベース・システムが含むべきすべての要素を網羅したものではなかった．しかしこの共通言語体系を基本としたシステムは明らかに共通のはっきりしたイメージをもっているので，本稿ではこれを CODASYL 方式のデータベース・システムと呼ぶ．

共通言語体系がまとまりはじめたころから，この方式についての賛否の議論もさかんに行われた．最近のわが国においても，「日本のデータベース関係者は CODASYL 方式に毒されいている」という見方から，「アメリカの学会ではもはや CODASYL 方式についての論文は出なくなった．あれは死に絶えた」という意見まで評価の振幅がおおきい．現実には，FORTRAN や COBOL に関する論文が学会に出なくなったからといって，これらをだれも使わなくなったわけではないし，各計算機にコンパイラがあるからといって，これらをだれもが満足しきって使っているわけでもない．それは現時点における現実的な道具としてさかんに使われているのである．

CODASYL 方式のデータベース・システムの特徴は，

（1） データベース用共通言語体系であること．

（2） 現在時点におけるソフトウェア技術水準で実現可能なまとまった方式であること．

（3） IDS の影響を強く受けていること．

などであろう．

本稿ではこれを，（1）システム環境，（2）データ構造，（3）データ定義とデータ操作，（4）開発の歴史の四つの方向から分析解説する．またそれぞれのおもな問題点を取り上げて，今後のデータベース・システム研究上の問題提起を行う．

なお CODASYL のデータベース用共通言語体系の仕様書はすでに公刊されている[3),4)]ので，本稿ではこれらを中心に解説した．近い将来改訂がきまっている事項のうち重要なものについては，そのむねを明記して言及した．

1. CODASYL 方式のデータベース・システム環境

1.1 概　説

CODASYL 方式のデータベース・システムの概念を**図-1** に示す．ここでデータベースとは，これまでは応用プログラムごとにばらばらに作成維持してきたファイル群を効率よく一つに統合したものをいう．ふつうは企業，大学，研究所，病院などの組織体 (enterprise) のレベルでの統合を考える．さまざまの目的を持つ利用者たちがデータベースを共有し，場合によっ

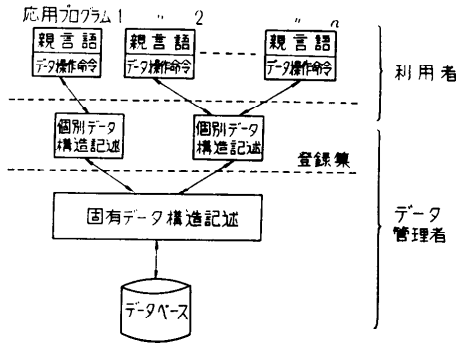


図-1 CODASYL 方式によるデータベース・システム[17)]

* Database Systems based on CODASYL Approach by Syunsuke UEMURA (Computer Science Division, Electrotechnical Laboratory)

** 電子技術総合研究所ソフトウェア部言語処理研究室

ては同時に利用する．データベースそのものを含めてデータベース・システム全体を管理するのは，データ管理者（DA；Data Administrator, DBA すなわち Database Administrator も同義）と呼ばれるソフトウェア専門家（の集団）である．データ管理者は強力なデータ記述言語を使ってデータベース全体を設計管理する．データベース全体の記述を固有データ構造記述（schema, スキーマ）という．データ管理者は固有データ構造記述言語（Data Description Language for the Schema, スキーマ DDL）を使用する．このデータ記述言語の言語仕様[3]が，CODASYL のデータベース用共通言語体系の第1の要素である．固有データ構造記述言語によって記述したデータベースは，現行のあらゆる高水準プログラム言語から独立しており，しかも共通して操作可能であるように配慮されている．

データベースの利用者はそれぞれのプログラム言語を介してデータベースを操作する．このために，個々の応用プログラムの側から見たデータベースの記述と，データベースを操作する命令とが必要になる．前者を個別データ構造記述（subschema, サブスキーマ），後者をデータ操作言語（DML；Data Manipulation Language）という．個々の応用プログラマは，全データベースのうち自身の作業に必要な部分のデータ記述だけを知っていればよい．しかしそれは応用プログラムを書くプログラム言語になじむ形式のデータ記述でなければならない．データ操作言語も応用プログラムを書くプログラム言語となじむ形式でなければならない．したがって，これらはプログラム言語ごとに最適の言語仕様が定められるべきである．このために COBOL には一群のデータ操作命令と，個別データ構造記述言語の仕様とが追加された[4]．これをまとめて COBOL データベース機能と呼ぶ．データベースへの COBOL インタフェースとも考えられる．これが CODASYL のデータベース用共通言語体系の第2の要素である．

さらに CODASYL では，FORTRAN の言語仕様にデータベース機能を追加して，FORTRAN インタフェースを実現する作業を進めている[5]．

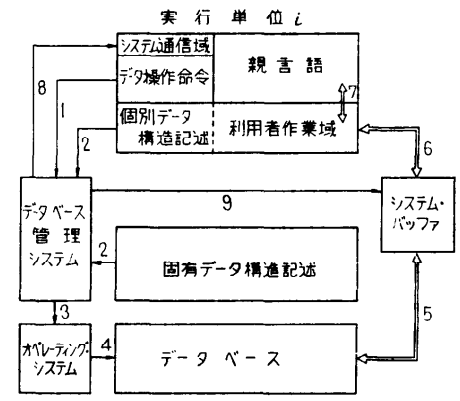
これらの要素をまとめて，CODASYL のデータベース用共通言語体系と総称する．

データベース機能を追加するもとになったプログラム言語（ここでは COBOL や FORTRAN）を親言語（host language）という．データベースを直接操作する作業はデータベース機能が担当する．データベースから取り出したデータの加工は親言語が担当する．このような方式を親言語方式という．現在の CODASYL 方式のデータベース・システムは原則として親言語方式である．したがってそれは，いわゆるプログラマを利用者として想定している．

1.2 ソフトウェア・システム構成

すでに述べたように，CODASYL のデータベース用共通言語体系は，データベース・システム全体の仕様を完全に規定したものではないが，これにもとづく典型的なデータベース・システムのソフトウェア構成はたとえば図-2 のように想定される[3]．

図-2 において実行単位（run unit）とは，一つ以上のプログラムの実行をいう．実行単位はデータ操作命令によってデータベースのデータ操作を要求する(1)．データベース管理システムがこの要求を受け取り，データベースの固有データ構造記述と，実行単位が参照している個別データ構造記述とを参考にしながら必要な情報を補う(2)．これらの情報をもとにして，データベース管理システムはオペレーティング・システムに対して物理的な入出力命令を要求する(3)．オペレーティング・システムは記憶媒体を操作して(4)，データをシステム・バッファへ転送する(5)．データベース管理システムは必要におうじてシステム・バッファのデータを実行単位の利用者作業域に移す(6)．このとき固有データ構造記述と個別データ構造記述との間でのデータの変換は自動的に行われる．利用者作業域は個別データ構造記述にしたがって設定される領域で，親言語の命令はこの領域のデータを自由に操作できる(7)．データベース管理システムは要求の実行結果などの情報を利用者のシステム通信域に与える



（→は制御の流れ，⇒はデータの流れを示す）

図-2 ソフトウェア・システム構成と流れ

(8). データベース管理システムはまたシステム・バッファの管理をも行う(9). 実行単位はシステム・バッファを通じてだけデータベースと連絡できる.

実際にはデータベース管理システムが複数個の実行単位を同時に処理することもある. しかしその「同時に」の明確な定義は, オペレーティング・システムの定義と同様に, 共通言語の仕様の範囲外とされている. 現在はデータベース用共通言語の範囲内に入っていないが, 完結したデータベース・システムを構成するために必要とされる言語仕様には以下のものがある.

データベースの日常業務を行うユーティリティ.

データベース回復ルーチン.

固有データ構造, 個別データ構造の修正を簡単に行える言語.

データを実際の装置や媒体空間に割り付けるための言語すなわち DMCL (Device Media Control Language).

これらの言語仕様の開発は CODASYL の将来の課題として残されている.

1.3 データベース・システム環境上の問題点

データベース・システム環境面では, 親言語方式である点と, 2層のデータ記述 (固有データ構造記述と個別データ構造記述) の概念とがとくに論争の的になった.

1971 年ごろは親言語方式であること自体が欠点であるかのような議論もあったが, 最近ではその現実的な利点が認識されて, この点についての異論はあまりみられない. 関係モデルにもとづく一連の実験的データベース・システムも, その頂点に立つと考えられる System R を含めて, 親言語方式がおおい.

しかし親言語の拡張として定義されるデータベース機能の部分が, あまりに手続き的であるとの批判はいぜんとして残っている. 現在の COBOL および FORTRAN データベース・インタフェースはすくなくともその親言語と同等の複雑さを持っており, プログラマでないと使いこなせない. 固有データ構造記述言語で記述作成したデータベースに, プログラマでない利用者 (end user, terminal user, casual user) 向きのインタフェースを設定することは可能であり, 現実に実動化が試みられている例もある. この場合には, この種のインタフェースを直接設定するか, COBOL のデータベース機能などを介して間接的に設定するかが問題であろう. CODASYL 内部でもこの種の問題を検討している作業班がある[7].

データベース・システムのデータ構造とその記述とを, データベースの全体像である固有データ構造 (スキーマ) と, 個々のプログラムから見た個別データ構造 (サブスキーマ) との2層構造としてとらえる概念は 1969 年の DBTG 提案から折り込まれていた. この概念はいくつかの優れた点を持っていたが, なおいわゆるデータ構造と記憶構造とが混在しているとの批判を受けていた. たとえばデータ管理者は固有データ構造記述を作成するにあたって, 現実の世界をモデル化する作業と, そのモデルを計算機の記憶構造に写像する作業とを同時に行わなければならない. いっぽうデータベース・システムの研究の進展にしたがって, 多層のデータ・モデルの概念が提起されてきた. その中でもっとも典型的な多層モデルとして評価が高いのは, アメリカ規格協会データベース研究グループによる3層スキーマの提案であろう[6),14)]. 図-3 はこの概念を示す. 外部スキーマは CODASYL 方式における個別データ構造記述に相当する. これまでの固有データ構造記述はここでは概念スキーマと内部スキーマとに分化している. 前者は現実世界のデータの論理的なモデルの記述で, データベースの全データの属性や相互間の関係を記述する. 後者はいわゆる記憶構造の記述で, 計算機上でポインタ方式で実動化するか, 表形式にするか, 索引や転置ファイルを準備するかどうかなどを記述する. CODASYL データ記述言語委員会 (**4.3** 参照) ではこの提案に注目して, 固有データ構造記述言語をこの方向で洗い直し, 言語仕様の機能的分類などの作業を行っている (**3.1.1** 参照).

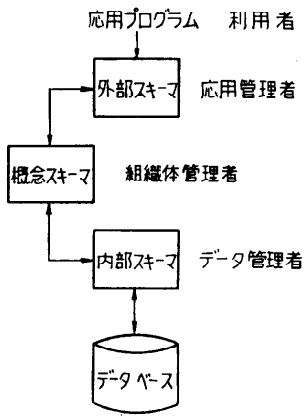


図-3 3層のスキーマ

2. CODASYL 方式におけるデータ構造

2.1 データ構造の表現

データベース・システムの特徴をよく示す要素の一つはそのデータ構造である．CODASYL 方式のデータベース・システムは IDS の創始者 C. W. Bachman によるデータ構造記述手法をそのまま踏襲している[8]．

このデータベース・システムには，まず COBOL のそれにほぼ相当するレコードの概念がある．レコードはデータ項目などの集まりに名前を付けたもので，入出力命令が1回に操作する基本単位である．

プログラム中にレコード記述を書くことによって，レコードの型 (type) が定義される．一つのレコード型に対して，データベース中には任意個の実現値 (occurrence) が存在しうる．たとえば KYUUYO-RECORD というレコードを記述すると，一つのレコード型が定義される．データベース中には，このレコード型の実現値が社員当り一人ずつ，すなわち社員の人数だけ存在するであろう．

レコード内のデータ構造は，COBOL 式のレベル番号や反復 (OCCURS) 句を使って記述する．このほかに，レコード相互間の関係を記述する親子集合 (set) の概念がある．

親子集合は，一つのレコード型を親 (owner)，一つ以上のレコード型を子 (member) と指定することによって定義される．親子集合にも型と実現値とがある．

以下では，実現値ということばを原則として省略する．たんにレコードまたは親子集合といえば，それは実現値を意味する．

C. W. Bachman はこれらの関係を**図-4** のように表現した[9]．図-4 において長方形はレコード型を，矢印は親子集合の関係を示す．長方形の中にレコード型の名前を，矢印のわきに親子集合型の名前を書く．このような図は創始者の名前をかりて，バックマン線図 (Bachman diagram) と呼ばれる．バックマン線図はレコード型の関係を表示し，実現値の関係はなにも示さない．T. W. Olle[12] や G. M. Nijssen[16] は実現値の関係の表示法を提案している (**図-5** 参照)．

図-4 親子集合型（バックマン線図）

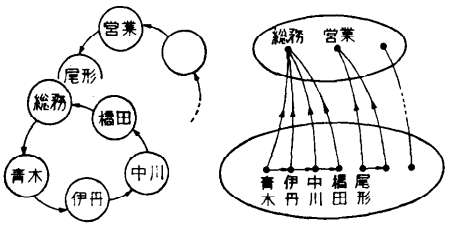


図-5 実現値の表示（左 Olle, 右 Nijssen による）

2.2 表現能力と制限事項

親子集合を構成するおもな規則は以下のとおりである．

（1） 親子集合型は一つの親レコード型と一つ以上の子レコード型とからなる（後述の特殊な例外もある）．

（2） 固有データ構造記述中には，任意個数の親子集合型を宣言できる．

（3） 一つのレコード型を，一つ以上の親子集合型の親として宣言できる．

（4） 一つのレコード型を，一つ以上の親子集合型の子として宣言できる．

（5） 一つのレコード型を，一つ以上の親子集合型の親であり，かつそれとは別の一つ以上の親子集合型の子であるとして宣言できる．同じ親子集合型の親でありかつ子であると宣言することはできない．

（6） 親子集合には親レコードがただ一つだけ存在する．

（7） 一つの親子集合型内では，子レコードは複数の親子集合には参加できない．すなわち一つの親子集合型内では，子レコードはただ一つの親レコードにだけ対応する．

以上の規則から，この方式のデータ構造表現能力や制限事項が明らかになる．おもなものを列挙する．

（a） 規則(1)が基本的な親子集合型を定める．単一の親子集合型はレコード型のレベルで木構造 (tree) を表現する．すなわち親レコード型と子レコード型とは1対 n 対応をなしている．さらに規則(3),(4)によって，一つのレコード型を複数の親子集合型の親や子として重ねて宣言できるので，これによってレコード型のレベルで網構造 (network, graph) を表現することも可能になっている (**図-6** 参照)．

（b） 規則(1),(6),(7)により，単一の親子集合において親レコードと子レコードとは実現値のレベルでも1対 n 対応をなしている．規則(7)は親レコードと子レコードとの間での n 対 m 対応を陽に禁止してい

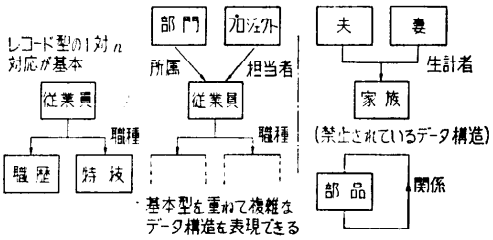


図-6 データ構造の表現と制限

る．すなわち一つの親子集合型に着目すれば，親レコードが決まれば，これに属するn個の子レコードが決まり，子レコードが決まれば対応する親レコードは一意に決まる．実現値のレベルにおいて親と子とのn対m対応を実現させる手段として，構造レコード (structure record)，関係レコード (relational record) などの手法がよく知られている（図-7 参照）．

（c） 規則(5)によって，レコード型のレベルにおける直接的な輪 (cycle) 構造は禁止されている（図-6 参照）．ただし親子集合型宣言を重ねていくことによって自然にできる間接的な輪構造は禁止されていない．この場合には，レコード型の属性に制限があるが詳細ははぶく．

なお2種類の特殊な形態の親子集合が規定されている．特異親子集合 (singular set) は，親レコード型をシステム (SYSTEM) と宣言した親子集合で，この親子集合の実現値はデータベース中に一つしか存在しない．これは順編成ファイルの記述に利用できる．動的親子集合 (dynamic set) は，親レコード型だけを宣言して，子レコードの設定は実行時のデータ操作命令にまかせてしまうものである．データ操作命令の実行によって，任意のレコード型のレコードがこの親子集合の子レコードになりうる．

以上の議論から，CODASYL 方式を一口に網方式と呼ぶのがあまりに安易であることは明らかであろう．また網とか階層（木）とかはデータ構造を表す用

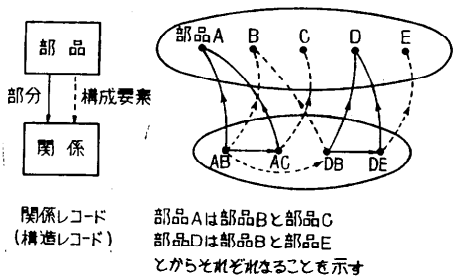


図-7 関係レコードによる実現値のn対m対応

語としてすでに定着している．したがって，階層方式 (IMS)，網方式 (CODASYL)，関係方式（関係モデル）といった対比[17]はあまり意味がないし，かえって誤解をまねきかねない．

2.3 データ構造表現手法に関する議論

親レコードと子レコードとからなる親子集合の概念は，現実社会にしばしば見られる階層（木）構造を自然に反映している．

CODASYL では，DBTG 提案（4.2 参照）時代からこれを set と呼び，プログラム中でも SET と宣言する．データベースの関係モデルの登場とともに，関係モデルにおける数学的に厳密な集合 (set) の取扱いと区別する意味で，CODASYL 方式のそれを owner-coupled set[16], coset[15], fan set[18] などと呼ぶことが行われ始めた．バックマン自身はこれを date-structure-set[11] と呼んでいる．本稿では親子集合という訳語を採用した．これらはすべて同じ概念を指す用語である．

CODASYL 方式と関係モデル方式との相違を示す例を図-8 にあげる．両者を比較する上での着眼点をいくつか以下に列挙する．

（1） CODASYL 方式では，親子集合の概念（図では「所属」）がデータ構造とデータの論理的な呼出し経路 (access path) とを陽に示している．これを情報搬送構造 (information bearing structure) とも呼ぶ[14),16]．一方関係モデル方式では，部門，従業員という二つの関係を結び付ける情報（図の「部門番号」）が両方に独立して存在するだけで，従業員の所属部門を示すようなデータ構造は陽には存在しない．なお関係の概念そのものをさらに分化して，情報搬送構造に対応する性質の関係 (relationship relation) を設定する手法も提案されている．

（2） 「所属部門」という情報は本来従業員固有の属性である．CODASYL 方式では，従業員の属性のうちこれだけをデータ構造的に表現し，他の属性はデータとして表現しており，不統一

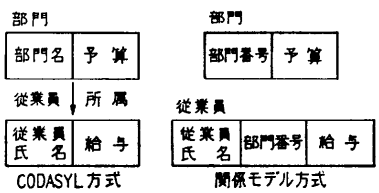


図-8 CODASYL 方式と関係モデル方式[16),18]

である．CODASYL 方式でもすべての属性をデータとして表現することは可能ではあるが，ある事象の属性を構造的にもデータ的にも表現可能というのは，両刃のやいばである．

（3） 関係モデル方式では，従業員1人ずつが部門番号をデータとして保持しており，冗長かつ不自然である．

（4） 関係モデル方式では，関係相互間の論理的なつながりを利用者が問い合わせ時に指定する．これは利用者自身がデータの論理的な呼び出し経路（データ構造）を定義しつつ問い合わせを行っていると考えられる．

（5） 親子集合の概念を実動化する手法として，チェーン（リスト構造），転置ファイル（ポインタ配列）などがある．言語仕様に実動化手法が規定されているわけではないが，過去のいきさつを反映して，親子集合にはなおリスト構造特有の制限事項が生きており，またデータの論理的な呼出し経路と物理的な呼出し経路とが完全には分離されていない．

3. データ定義とデータ操作

3.1 固有データ構造記述言語

3.1.1 言語のおもな機能

CODASYL のデータベース用共通言語体系における固有データ構造記述言語はコンパイラレベルのデータ記述言語である．この言語はデータ記述の機能だけで完結している．

データ管理者はこの言語を使ってデータベース全体を記述する．この言語の記述項に含まれる各句は機能的にみて以下の九つ（現状では八つ）に類別される．これによって，この言語の機能面の概要，ひいてはデータ管理者がこの言語を通して行う職務が明らかになろう．さきの3層スキーマの概念（1.3 参照）との対応づけも可能になる．

（1） 固有データ構造 (schema)——固有データ構造記述すなわちデータベース全体を記述したプログラムに名前をつけ，その性質を宣言する．SCHEMA NAME 句など．

（2） 構造 (structure)——データ構造とその構成要素との型を宣言する．データベースデータ名，OWNER 句，MEMBER 句，RECORD NAME 句，SET NAME 句など．

（3） 正当性の確認 (validation)——前項の構造類で宣言したデータ構造の，具体的な実現値のレベルでの制限事項を宣言する．すなわちデータベース中の個々のデータ値の正当性を確認する．CHECK 句，PICTURE 句，TYPE 句など．

（4） データ操作インタフェース (DML interface)——実際のデータ操作機能実行にあたって呼び出される手続きや，それに与えるべきパラメータを宣言する．SELECTION 句，WITHIN 句など．

（5） 呼び出し制御 (access control)——データベース中のデータ値を呼び出したり変更したりする場合の，利用者の資格を限定する機構を宣言する．PRIVACY 句 (ACCESS-CONTROL 句と名称が変る予定)．

（6） 計測 (measurement)——データベース管理システムに指示して，データベースの利用状況などに関するデータを集めさせる．これに相当する句はまだない．

（7） 性能向上 (tuning)——データベース・システムとしての性能を向上させるための手法を宣言する．LINKED 句，LOCATION 句，ORDER 句など．

（8） 資源割当て (resource allocation)——システムの資源管理に適切な単位を宣言し，データ構造実現値のそこへの割当てを制御する，AREA NAME 句，WITHIN 句．

（9） 管理 (administration)——データベース管理者が提供する手続きに名前を付け，それを呼び出す．CALL 句，ENCODING 句．

3.1.2 親子集合の属性

親子集合の属性のうちおもなものに簡単にふれる．

親子集合の親レコードと子レコードとの間のつながりの強さを示す属性を親子関係 (set membership) 属性という．つながりが強い子レコードはデータベースに格納されると自動的に親子関係が達成され(AUTOMATIC)，いったん達成された親子関係を切り離すことはできない (MANDATORY)．つながりが弱い子レコードはデータベースに格納されただけでは親子関係が成立せず，相当するデータ操作機能の実行によってはじめて達成される (MANUAL) し，さらにデータ操作機能によって親子関係を切り離すこともできる (OPTIONAL)．なお MANDATORY よりさらに強い FIXED 属性の導入も予定されている．

つぎに，親子集合型ごとにその子レコードが並ぶ順

序を示す順序づけ (ordering) 属性がある．一つのレコード型が複数個の親子集合型の子レコード型になっている場合には，それぞれに異なる順序づけを指定できる．データベース管理システムは，利用者が子レコードを親子集合に関係づけるつど，指定された順序を自動的に保守する．レコード中のキー項目の値による順序 (SORTED)，親レコードの直後，直前 (FIRST, LAST)，現在参照しているレコードの直後，直前 (NEXT, PRIOR)，データベース管理システムまかせ (SYSTEM-DEFAULT，予定) の四種類の順序づけ指定がある．

なお，親子集合における子レコード呼出しの能率の向上のために，子レコード型中の任意個のデータ項目を探索用キー (SEARCH KEY) として指定できる．また順序を SORTED と指定した子レコードに対してはさらに索引つき (INDEXED) と指定できる．これらの指定を行うと，データベース管理システムは目的の子レコードを能率よく探索できるようななんらかの手段，たとえば索引や転置ファイルを自動的に準備し，維持する．

3.1.3 レコードの一意な識別

データベース管理システムはデータベース中のすべてのレコード実現値を一意に識別できなければならない．いっぽうデータベースの利用者はレコード中のなんらかの属性(たとえば社員番号)を手がかりとして，特定のレコード型の範囲内でその実現値を一意に識別できれば十分なことがおおい．

現在の固有データ構造記述言語では，データベース中のすべてのレコード実現値を一意に識別するデータベースキー (DATA-BASE-KEY) の概念があるが，この言語仕様を改訂して，データベースキーの概念を全面的に削除することがすでに予定されている．そのかわりに，特定のレコード型中のレコード実現値を一意に識別する識別子 (IDENTIFIER) の概念が導入される．この識別子は対象とするレコード型に含まれるデータ項目 (の組合せ) である．関係モデルにおける主キーに相当する概念であろう．

COBOL インタフェース (次項参照) でもこれに応じた改訂を行う方針がすでにきまっている．

3.2 データベース機能

3.2.1 COBOL インタフェース

CODASYL COBOL, 1976[4] (4.3 参照) には，これまでの COBOL の拡張として，個別データ構造記述言語とデータ操作命令との文法が組み込まれている．これらと固有データ構造記述とが互いにかかわり合いながらデータベース用共通言語体系が動作する様子を図-9 に示す．

データベースのデータ構造とその構成要素の定義は固有データ構造記述によってほとんど固定され，個別データ構造記述はただその一部分を参照するにすぎない．データの基数など一部の属性は両方の記述が異なっていてもよく，この場合はデータベース管理システムが変換を自動的に行う．一つの固有データ構造記述に対して，個別データ構造記述が任意個存在してもよく，これらが互いに重複していてもよい．データ管理者は COBOL 個別データ構造記述登録集 (ソースプログラム・ライブラリ) をあらかじめ作成しておく．

利用者は各自の作業に必要な個別データ構造記述を COBOL プログラムのデータ部に指定して，この登録集から呼び出す．この段階ではデータ管理者との間で協議が必要であろう．場合によっては，新たな個別データ構造記述を生成することが必要になるかもしれない．

個別データ構造記述はデータベースの部分の記述であるとともに，応用プログラムにおけるデータベース用入出力領域の定義でもある．個別データ構造記述にもとづいて用意される入出力領域を利用者作業域 (UWA ; user working area) という．応用プログラムはこの領域を自由に参照でき，この領域を介してデータベースと接触する (図-2 参照)．

応用プログラムの実行が始まると，そのプログラムがデータベース中でいちばん最近に参照したレコードを表示する現在指示子 (currency status indicator) が自動的に動作し始める．現在指示子は概念的なポインタで，これが指すレコードを現在レコード (current record) という．現在指示子は 4 種類あり，各領域の現在レコード，各親子集合の現在レコード，各レコード型の現在レコード，各実行単位の現在レコードをそれぞれ指している．あらゆるデータベース操作は現在レコードを中心に進められるので，利用者はなにが現在レコードであるかをつねに明確に意識していなければならない．

COBOL に追加されたデータ操作命令を図-9 にあげた．データ操作命令の実行結果は新設された DB-STATUS などの特殊レジスタに表示されるので，応用プログラム自身でこれを確認できる．

3.2.2 FORTRAN インタフェース

FORTRAN インタフェースはアメリカ規格 FORT-

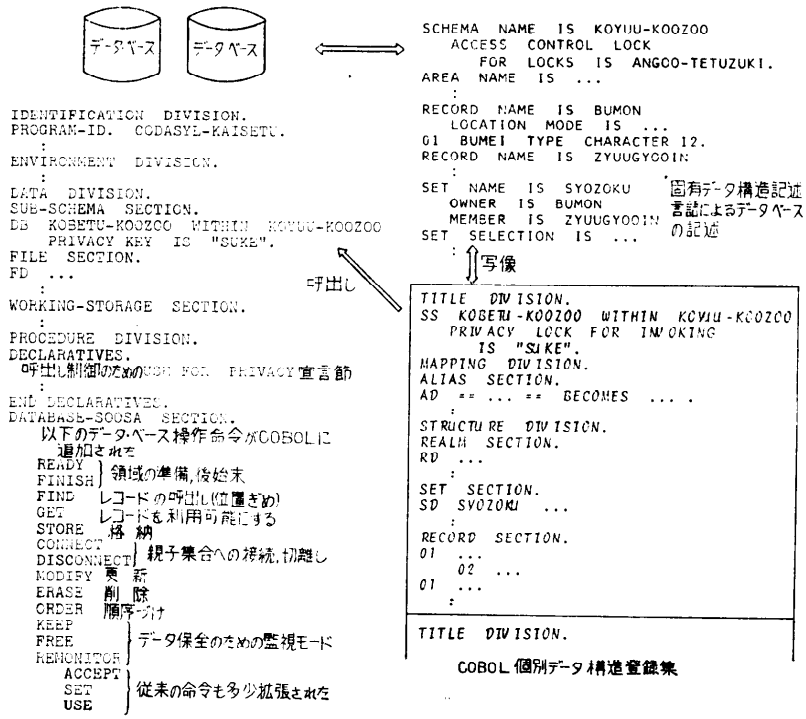


図-9 データベース用共通言語体系の動作の流れ (COBOL インタフェースの場合)

RAN の改訂案 FORTREV を基礎として，その拡張として計画されている (**4.3** 参照).

FORTRAN インタフェースは COBOL インタフェースをそっくりそのまま FORTRAN に移し変えたものと考えられる．個別データ構造記述とその登録集，データ操作命令などの概念はそのまま生きている．

データ操作命令は FORTRAN のサブルーチン呼出し形式ではなくて，新しい一群の命令として設定されている．FORTREV におけるキーワード方式によるパラメータ指定を多用している．命令の意味内容は COBOL のそれとほとんど同一で，ほぼ1対1に対応している．必要な個別データ構造記述を呼び出す INVOKE 命令，FIND 命令と GET 命令とを結合した FETCH 命令なども準備されている．

データ操作命令に関して注目されるのは，それが参照するレコード型名，親子集合型名などを変数の値として指定できる機能の導入である．これによって，命令が参照するデータ名を実行時に決定できる．

FORTRAN インタフェースの例を**図-10** にあげる．

3.3 データ記述とデータ操作の分離と干渉

3.3.1 データ独立

データベースの物理的，論理的な構造に変更があっても，それを操作する手続き（応用プログラム）は変更しなくてもすむようにしたいというのがデータ独立 (data independence) の概念である．CODASYL 方式のデータベース・システムはこのような概念の成熟を待たずに設計が終ってしまったので，データ独立を欠くとの批判をしばしば受けている．

データ独立を達成するために CODASYL 方式が採用したのは，データベースの定義とその操作とを分離

```
      PROG  REIDAI
      INVOKE(SUBSCHEMA=KOBETU,SCHEMA="FORTDB")
      ISUM=0
      SUMSAL=0
      READY(REALM=BUBUN1)
      FETCH(FIRST,RECORD=REC1)
   10 IF(DBSTAT .EQ. 1602400) GO TO 999
      SUMSAL=SUMSAL+SALARY
        :
```

```
SUBSCHEMA KOBETU, SCHEMA="FORTDB"
ALIAS(RECORD) REC1="DBREC1"
REALM BUBUN1
SET ENG
RECORD REC1(CHECK=DATA TYPE)
INTEGER SALARY
END
```

図-10 FORTRAN インタフェースの例

するという方向であった．固有データ構造記述だけを独立させたのはこの例である．さらに応用プログラムはそれぞれの仕事に最低限必要なデータベース部分の記述（個別データ構造記述）だけを通してデータベースと接触するので，データベースのこれ以外の部入の変更には影響されなくてすむ．

この方向を追求してきた結果すくなくとも以下の問題点が明らかになってきたと考えられる．

（1） データ定義とデータ操作を分離してデータ独立を達成する方向には本質的な無理があるのではないか．固有データ構造記述言語の中でも，具体的なデータ操作機能に言及せざるをえない部分がある．またデータベース操作機能はどうしてもデータ構造を参照するが，データ構造はデータ定義時にほぼ固定されてしまうので，両者を完全には分離できない．

（2） どのような方向をとるにせよ，完全なデータ独立を達成することはおそらく不可能である．「データ定義に変更があってもデータ操作手続きの変更は最小限にとどめる」というほうがより現実的である．

（3） データ独立とシステム能率（の低下）との間には密接な関係がある．

3.3.2 レコードの格納，配置，呼出し

固有データ構造記述言語には，レコードをデータベースに格納するときの配置手法を指定する配置（LOCATION）句があり，データ管理者がレコード型ごとにこれを指定する．以下の三つの配置手法がある．

（1） 親子集合型の子レコードがなるべく近くにまとまるように配置する（VIA SET）．

（2） レコード中のデータ項目をキーとして，その値を変換して配置場所をきめる（CALC）．

（3） システムまかせ（SYSTEM-DEFAULT）．

実際のレコードの格納や呼出しにあたっては，まず親子集合の実現値の選択が行われる．これは固有データ構造記述言語と個別データ構造記述言語の両方にある選択（SELECTION）句によって指定する．両方に指定があると，個別データ構造記述言語のそれが優先される．親子集合実現値の選択は通常なんらかの手段で親レコード実現値を順次決定していくことによって行う．以下の二つの選択手法がある．

（1） 現在選んでいる実現値をそのまま選択する．

（2） 親レコード中のデータ項目の値を手がかりとして選択する．

レコードの呼出しを行う FIND 命令では，レコード呼出しの手がかりとして，以下の指定が可能である．

（1） 配置時に使用した CALC キー項目の値．

（2） 親子集合の現在のレコードの直前，直後．

（3） 親子集合の最初，最後，n 番目，親レコード．

（4） 領域中の相対的な位置．

これらの指定をうまく組み合わせて，システムとしての能率を向上できる．その反面データ独立に好ましくない影響が現われうる．たとえば，配置句と FIND 命令の書き方との組合せに制限があり，レコードの配置手法が呼出し手法に影響している．

固有データ構造記述言語の例では，この配置句や，LINKED 句，SEARCH 句など性能向上を目的とする言語要素はいったん現在の言語仕様から削除しつつある．しかしこの種の機能の必要性は認めており，今後なんらかの形で再登場すると考えられる．

3.3.3 選択（SELECTION）句の機能拡張

これまでの言語仕様では，子レコードの属性値をもとにして親子集合実現値を選択するという機能がきわめて貧弱であった．前述のレコード型内のレコードの一意識別子導入に呼応して，親レコードの一意識別子（たとえば課名）と子レコードの属性値（たとえば所属する課名）とを手がかりとした親子集合実現値選択の機能の導入が予定されている．これは選択句の機能の大拡張である．

3.4 データ呼出し資格の制御

データベース中のデータを資格のない人による呼出しから保護する機能が用意されている．これまではプライバシ（PRIVACY）句と呼ばれてきたが，固有データ構造記述言語ではこれを呼出し制御（ACCESS-CONTROL）句と名称変更する予定である．

データ管理者はデータ構造のそれぞれのレベルに想定されるデータ操作ごとに，呼び出し制御ロックを宣言する．応用プログラムがこれらのデータ操作を行うためには，適切な呼出し制御キーを与えてシステムの許可を得なければならない．ロックまたはキーとして使用できるのは，文字直定数，データ項目，手続きのいずれかである．

たとえば固有データ構造記述中で，

```
01 KYUUYO ACCESS-CONTROL LOCK FOR GET
        IS "KYUUYO-GAKARI" …
```

と宣言されている KYUUYO レコードに対して，GET 命令を実行させたい利用者は，その COBOL プ

ログラムの宣言節中に

```
KEY-SETTEI SECTION.
    USE FOR PRIVACY ON GET FOR KYUUYO.
DB-AC-KEY-KYUUYO.
    MOVE "KYUUYO-GAKARI" TO
    DB-PRIVACY-KEY.
```

と宣言しておかなければならない．DB-PRIVACY-KEY は特殊レジスタで，この宣言節は領域を開く時点で実行される．

さきの図-2 にもあるように，データベース管理システムとオペレーティング・システムとは密接に関係しているので，このような呼出し制御機能はオペレーティング・システムによって骨抜きにされる可能性がある．ただしデータベース・システムにおける呼出し制御はオペレーティング・システムのフィイル管理等におけるそれよりかなりきめが細かい．

なお個別データ構造記述でも呼び出し制御指定ができる．両方の指定が重なると，個別データ構造記述のそれが優先される．これは，

（1） 個々の個別データ構造記述を別個に制御でき，

（2） 利用者ごとにきめ細かく制御指定できる，

などの利点を持っている．

また呼出し制御ロックの管理は固有データ構造記述言語とは別個の言語体系にまとめるべきだとの議論がある[12]．

3.5 データ保全

COBOL データベース操作機能には，同時実行単位によるデータベース同時更新によって，データの内容が無意味にならないかどうかを管理する機能がある．この機能は，特定の命令 (MODIFY, CONNECT, DISCONNECT, ERASE) の実行によってレコードが現在レコードになると動作し始める．その時点でレコードは監視モード (monitored mode) に入り，現在レコードでなくなるとこのモードでもなくなる．KEEP 命令と FREE 命令とによって，監視モードを拡張することもできる．このモードにあるレコードはシステムの監視をうけ，同時実行単位による変更が行われなかったかどうか自動的に検査される．同時実行単位による汚染が検出されると，命令の実行は行われずに，誤り状態だけが表示される．

データ保全の確保をこのように利用者まかせにすることは従来から問題になっていた[19),20)]．COBOL データベース機能では，データ保全はシステムまかせにする方向で改訂を検討中である．方針として，利用者がある領域を更新モードで開く (READY) と，その利用者はこの領域から呼び出すレコードを専有する方向のようである．現時点ではこれより方法がないのであろうが，デッドロック増加のおそれがある．

4. データベース用共通言語体系の開発

4.1 CODASYL の活動

CODASYL (The Conference on Data Systems Languages；データシステム言語協議会) は事務処理応用のための共通言語を開発する目的で 1959 年 5 月に設立された．CODASYL は計算機業者，利用者の協力による任意団体で，これまでに事務用共通言語 COBOL を開発したほか，情報代数 (Information Algebra)，決定表 (DETAB-X) などを手がけてきた．

CODASYL COBOL の言語仕様を最終的に決定するのは，プログラム言語委員会 (PLC；Programming Language Committee) である．COBOL にデータベース機能を追加しようという作業は大きな反響をまき越こしながら行われ，結果として COBOL を含めたデータベース用共通言語体系の概念に広がっていった．

4.2 データベース作業班 (DBTG)

1965 年 6 月のプログラム言語委員会（当時は COBOL language subcommittee といった）に，COBOL にリスト処理機能を追加したいという提案が行われて，これを検討するためにリスト処理作業班が設置された[1),3)]．リスト処理作業班は 1967 年 5 月には，「作業目標をより明確にするために」その名称をデータベース作業班 (DBTG；Data Base Task Group) と改めた．

1968 年 1 月にデータベース作業班は，最初の簡単な報告書「データベース操作のための COBOL の拡張」[1)] を公表した．この報告書は GE 社の IDS[8)] や GM 社の APL の影響を色濃く受けながら，COBOL のデータベース機能がどうあるべきかを論じたものであった．

情報処理学会では当時（昭和 43 年）ソフトウェア研究会がいちはやくこれを取り上げて分析を行った．その後のデータベース用共通言語開発は，この報告書が定めた大わくに忠実にそって行われたといっても過言ではない．たとえばこの報告書では，端末利用者の重要性は認めながらも，まず応用プログラマ向きの水準に着手すべきであることを明言している．

1969 年 10 月にデータベース作業班は，「データベー

ス作業班から CODASYL プログラム言語委員会への報告 1969年10月」をまとめた．これには具体的な言語仕様が含まれていた．この報告書に対して，1971年までに200以上の提案，意見がデータベース作業班に寄せられ，審議された．

1971年4月にデータベース作業班は，「データベース作業班からプログラム言語委員会への報告 1971年4月」[2]を発表した．ふつうこの報告書を DBTG 提案，DBTG '71 などと呼ぶ．プログラム言語委員会は同年5月にこの報告書を審議して，基本線としてこれを受け入れることを承認した[28]．DBTG 提案には，1968年にかかげた長期目標のうちかなりの部分がまがりなりにも実現されていた．すなわちデータ定義とデータ操作との分離，プライバシ保護機能，個別データ構造の概念の導入などである．

1969年，1971年の報告書は，アメリカ国内のみならずヨーロッパでも広く注目を集めた．IBM 社の委員がこの報告書に積極的に反対したために，ジャーナリスティックな話題も集めた．日本の関心だけは低調のままに終始した．

4.3 データ記述言語と COBOL データベース機能

CODASYL とそのプログラム言語委員会は DBTG 提案を審議して，COBOL に直接関係する部分とそうでない部分とを分離することとした．すなわち DBTG 提案のうち，「固有データ構造（スキーマ）のためのデータ記述言語」の部分は独立したプログラム言語として処理する．このために CODASYL は新たにデータ記述言語委員会 (DDLC ; Data Description Language Committee) を設立した．同委員会によるデータ記述言語の最初の報告書は，CODASYL データ記述言語開発報告 1973年6月[3]というもので，1974年1月にアメリカ国立標準局から公刊された．これが固有データ構造記述言語の仕様書で，しばしば DDL JOD と略称される．

また DBTG 提案のうち，「COBOL 個別データ構造（サブスキーマ）のためのデータ記述言語」と「COBOL データ操作言語」の部分は，COBOL の拡張としてこれまでどおりプログラム言語委員会が担当することになった．同委員会は，DBTG 提案の相当部分を COBOL 言語仕様書の書式に合わせて書き直す作業を行い，1975年には最終的に文法を承認した．こうして，CODASYL COBOL 開発報告 1976[4] にデータベース機能が COBOL の正式の文法の一部として登場した．

この段階までは両者ともに DBTG 提案の基本線をなるべくくずさないように留意して作業を進めてきた．今後は，かつての COBOL がたどったのと同様に，大胆な変ぼうをとげるものと予想される．現在の言語仕様の主要な部分が全面的に変更になることはほとんどありえないであろうが，すでに述べたようにデータベースキーの概念を消してしまう程度の変更は十分にありうる．

なおプログラム言語委員会には，1974年に FORTRAN データ操作言語委員会ができて，1976年中に FORTRAN データベース機能を完成するべく作業を急いでいる[5]．

4.4 データベース用共通言語の標準化

CODASYL によるデータベース用共通言語の開発は，それがデータベース用言語の標準化に直結するものではないかという意味でも注目を集めた[28]．

1973年には，オランダから ISO（国際標準化機構）に対して，CODASYL 方式を基礎としてデータベース・システムおよび言語の標準化を審議すべきであるとの提案が行われた．これに呼応して，アメリカ規格協会にはデータベース研究グループが，また ISO/TC 97/SC 5（プログラム言語）にも同様のグループが設置された．後者のこれまでの結論では，現在の CODASYL 方式は標準化の対象となるには不十分であるとされている．アメリカ規格協会の研究グループによるインタフェース研究[6]（**1.3** をも参照のこと）のうち，応用プログラムインタフェースとして COBOL インタフェースをまず標準化する可能性はなお残っている．

5. おわりに

CODASYL 方式のデータベース・システムの実動化はきわめて盛んに行われている．すでに DBMS 2200 (NEAC 2200), ADBS (ACOS 4), AIM (M), EDMS, DMS-5 (MELCOM-COSMO), IDS-II (HIS, ACOS 6), DMS 1100 (UNIVAC 1100), DMS 90 (OUK 9000), DBMS-10, DBMS-11, DBMS-20 (DEC), IDMS（各社機種あり），SAAB DMS (SAAB) など十指にあまる商品が完成または開発途上にある．

CODASYL 方式はデータベース・システムの研究開発に一時代を画した金字塔であり，すくなくとも1970年代前半において技術的にもっともよく検討されまとまったシステム方式であった．今後は，データベース・システム研究の急展開を反映して，さらに息長く維持

改良されて行くことであろう．計算機社会にどのように受け入れられ，位置づけられていくかが興味深い．

参考文献

1) Report to the CODASYL COBOL Committee January, 1968 COBOL Extension to Handle Data Bases, Prepared by Date Base Task Group (1968).
2) Data Base Task Group Report to the CODASYL Programming Language Committee April 1971, ACM (1971).
3) CODASYL Data Description Language Journal of Development June 1973, NBS Handbook 113 (1974).
4) CODASYL COBOL Journal of Development 1976, Canadian Government 110-GP-1d (1976).
5) CODASYL FORTRAN Data Base Facility Version 0.5, 内部資料 (1976).
6) ANSI/X3/SPARC Study Group on Data Base Management Systems Interim Report 75-02-08, ANSI Doc. No. 7514 TS 01, ACM FDT (bulletin of SIGMOD), Vol. 7, No. 2 (1975).
7) A Progress Report on the Activities of the CODASYL End User Facility Task Group June 1975, ACM FDT (bulletin of SIGMOD), Vol. 8, No. 1 (1976).
8) C. W. Bachman & S. B. Williams: A General Purpose Programming System for Random Access Memories, Proc. AFIPS, Vol. 26 (FJCC, 1964).
9) C. W. Bachman: Data Structure Diagrams, DATABASE (ACM SIGBDP Newsletter), Vol. 1, No. 2 (1969).
10) C. W. Bachman: The Programmer as Navigator, Comm. ACM, Vol. 16, No. 11 (1973)—邦訳 bit 1974年12月号.
11) C. W. Bachman: Trends in Database Management-1975, Proc. AFIPS, Vol. 44 (NCC 1975).
12) Information Processing 74, Proceedings of the IFIP Congress 74, North-Holland (1974)—Olle (pp. 998～1006), Codd (pp. 1017～1021) など).
13) J. W. Klimbie & K. I. Koffeman (eds.): Data Base Management, Proc. IFIP TC-2 Working Conference, North-Holland (1974).
14) B. C. M. Douqué & G. M. Nijssen (eds.): Data Base Description, Proc. IFIP TC-2 Working Conference, North-Holland (1975)—Nijssen (pp. 1～71), Steel, Jr. (pp. 183～198) など.
15) G. M. Nijssen: Two Major Flaws in the CODASYL DDL 1973 and Proposed Corrections, Information Systems, Vol. 1, No. 4 (1975).
16) E. F. Codd & C. J. Date: An Interactive support for Non-Programmers The Relational and Network Approaches, 1974 ACM-SIGFIDET Workshop on Data Description, Access, and Control. (1974).
17) C. J. Date: An Introduction to Database Systems, Addison-Wesley (1975).
18) C. J. Date: An Architecture for High-level Database Extensions, 1976 ACM-SIGMOD Proc. (1976).
19) R. W. Engles: An Analysis of the April 1971 Data Base Task Group Report, 1971 ACM-SIGFIDET Workshop on Data Description, Access, and Control (1971).
20) R. W. Engles: Currency and Concurrency in the COBOL Data Base Facility, to appear in the Proc. IFIP TC-2 Working Conference (1976).
21) P. P. Chen: The Entity Relationship Model-Toward a Unified View of Data, ACM TODS, Vol. 1, No. 1 (1976).
22) M. Managaki, et al.: A Structural Model for System Implementation and its Application to CODASYL DBTG, 2nd USJCC Proc. (1975).
23) 植村俊亮: CODASYL の活動，情報処理第13巻第2号 (1972).
24) 植村，西村: CODASYL のデータベース用共通言語，情報処理学会データベース研究会資料 73-4 (1973).
25) データベースの諸問題，昭50電気四学会連合大会講演論文集第6分冊 S 41 (1975).
26) 植村俊亮: COBOL データベース機能，情報処理学会データベース研究会資料 DB 24-1 (1975).
27) 原潔: CODASYL 型のデータベース・マネジメント・システム—DMS 1100 について，情報処理学会データベース研究会資料 DB 24-2 (1975).
28) データベース言語研究委員会報告 1974～1975年度，情報処理第17巻第6号 (1976).

(昭和51年8月2日受付)