報　　告

汎用データベース操作システムの機能解析序論*

CODASYL システムズ委員会

この報告書は，CODASYL システムズ委員会で作成され，各委員の意見を表わしているが，必ずしも参加各社の見解を反映するものではない．CODASYL 理事会はこれをシステムズ委員会技術報告書として外部に発表することを承認した．

* *The Introduction* from *Feature Analysis of Generalized Data Base Management Systems*, by CODASYL Systems Committee, 8 December 1970, Chairman: T. William Olle RCA Corporation;
Members:
Gordon C. Everest　University of Minnesota (and Auerbach Corporation);
James P. Fry　University of Michigan (formerly with Mitre Corporation);
Mary E. Fuller　URS Data Sciences Company;
†Mary K. Hawes　Information Systems Leasing Corp.;
†Anthony J. Kay　Honeywell Information Systems;
Henry C. Lefkovits　Honeywell Information Systems (formerly with General Electric);
William C. McGee　IBM Corporation;
A. Metaxides　Bell Telephone Laboratories;
†Martin Rich　Esso Mathematics and Systems;
Richard F. Schubert　B. F. Goodrich Chemical;
Edgar H. Sibley　University of Michigan;
William H. Stieger　Chase, Brass and Copper Co.;
Alfred H. Vorhaus　MITRE Corporation (formerly with SDC);
Arla E. Weinert　Naval Command Systems Support Activity;
John W. Young　NCR Corporation;
† Resigned during 1970.

〔**訳注**　データ組織言語協議会 (CODASYL) には二つの委員会がある．プログラミング言語委員会は COBOL語の開発を活発に行なっている．学会が作成した「COBOL 1965 年版」，「JIS COBOL 原案」もまた，その活動に直接由来する。一方，システムズ委員会は，事務データ処理用共通システム開発の長期的見通しのもとに，以前に「情報代数」，「決定表」を発表し，最近はデータベースの調査研究にあたってきた．その第 1 次報告書[1]は，英語版，日本語版とも広く普及した．今回，その第 2 次報告書[2]が完成し，その配布を ACM，イギリス計算機学会，IFIP および当学会に依頼してきた．

当学会ではその日本語版を発行することを検討している．ここに掲載するのはその内容全体の紹介に相当する第 1 章序論の部分である．

なお，原報告書[2]の英語版については，洋書店などに依頼されたい．

CODASYL の方針として示されているとおり，この報告書の一部または全部を複製してもよいが，出所は明示されたい．

1. 前　史

1969 年 5 月に CODASYL システムズ委員会は「汎用データベース操作システムの調査」[1] を発表した．この報告書は種々のシステムを特徴づける機能を列挙し，そのあとに 9 個のシステムを章に分け，列挙された機能にしたがって記述した．その報告書は 1969 年 5 月 27 日，28 日にワシントンで開催された第 10 回 CODASYL 年会の参加者に配られた．

2. システムズ委員会の目標

システムズ委員会は次の段階として，統合されたデータベースシステムのための共通言語と機能の仕様を開発するという目標を追求した．標準システムの開発はアメリカ規格協会 (ANSI) の仕事なので，CODASYL での開発は“標準”ではなく“共通”という語が使われる．共通システムが広く作成され応用されれば，それが ANSI によって標準化されるであろうと期待している．COBOL が 1959 年から 1961 年のあいだに CODASYL によって開発され，1964 年に標準化のために ANSI (当時 ASA) に提出され，1966 年*1に ANSI 規格として採択されたと同じ経過を，データベースシステムもたどるであろう．

委員会は作業の基礎として，前の調査報告書を検討した．報告書中の記述を配列しなおして，各機能の記述ごとに各システムにおけるその機能の記述がつづくようにした．これを検討した結果，前の報告書における機能の記述はいささか不十分であって，包括的で完全な記述が必要であると思われた．それで，各機能についての文章記述を増補し，各システムでの機能の扱われ方をより完全に記述するようにした．

*1 1968 年制定

こうして第2次技術報告書が作られた．ここに掲載するのが，その第1章序論である．これは報告書本文の序論であるとともに，独立した正式文書として，雑誌などに発表されることをも目的としている．これはシステムズ委員会としての意見を表わしている．委員のあいだで意見が分かれて，討論がなされた事項は，論点として述べてある．それは賛否の両面を紹介するように努めた．

3. 報告書公刊の是非

汎用データベース操作システムの機能報告書を作成し，計算機社会に公表することの是非について，やや意見が分かれ，かなりの討論が行なわれた．多数の委員は，前の報告書だけでは委員会の長期目標である共通システムの設計への基礎としては不十分であって，2. のようにならべかえた構想のもとに，各機能のより詳細な記述を行なうべきであると考えた．

今回個々の商用システムについての情報を委員会が再び出版することについては，反対意見もあった．いろいろの機能の価値と役割を技術的に評価し方向を与えるために，これらのシステムを注意深く研究すべきであることは疑いない．しかし，各システムについての個別的な情報を調査し，出版することの価値については議論の余地があった．システムズ委員会が計算機社会にたいする調査サービスを提供するのだと思われたくはないという合意もあった．他方では，システムに触れないでその機能の一覧表をかかげるだけでは，技術報告書としては抽象的すぎるだろうし，システムの記述を含めることは報告書により多くの技術的内容を与え，汎用データベース操作システムの増大しつつある重要性により多くの注意を集めることになるだろうと思われた．システムの記述を含ませることによって，システム間の差違に注意を向けさせることも，重要だと思われた．

CODASYL COBOL を汎用データベース操作システムの調査のなかに含めることの是非も議論された．COBOL が汎用データベース操作システムの一種であるとは，一般にはみなされていない．しかし，主要なデータ構造をはじめとして，親言語システムや独立言語システムにみられる多くの機能は，まさに COBOL のなかにも含まれているのである．

なお，COBOL の維持開発は，システムズ委員会とならぶ組織であるところの，CODASYL プログラミング言語委員会の仕事であることを指摘しておこう．

4. この報告書の役割

この報告書の記述は，前の報告書よりも技術指導的になるように試みた．すなわち，各機能別に章を分け，章ごとにその機能を説明し，各システムにおけるその機能の状況を文章または一覧表で示した．とりあげたシステムは，CODASYL COBOL, DBTG の提案，IBM の GIS, GE の IDS, IBM の IMS, インフォーマティクス社の Mark Ⅳ, IBM の FFS (NIPS ともいう)，アウエルバッハ社とウェスタンエレクトリック社の SC-1, SDC の TDMS, RCA の UL/1 である．ある機能がすべてのシステムにわたって十分あきらかに確認され，かつ個々のシステムの能力について詳細な注をつける必要がない場合にかぎって，一覧表の形式をとった．文章による記述の詳しさは，すべてのシステムにわたってそろうように努めた．前の報告書では，この点が欠けていた．しかし，前の報告書のようにシステム別に章を分けるほうが，システムとしてのまとまった機能を記述するには適当であるということは指摘しておこう．一方，この報告書中の記述をシステム別に配列しなおしたとしたら，その結果は前の報告書ほどには有用でなくなるだろう．

この報告書の目的は，現在のシステムで用いられている機能をより完全に記述することである．この点で前の報告書の拡張になっている．さらに，この報告書で示した諸機能が共通システムを設計開発するさいの指針を与え，統合されたデータベースシステムのための共通言語と機能の仕様を開発するために使われることを期待する．

5. 技術の現状

現在，市場に出ている汎用データベース操作システムにはさまざまのものがある．名称としても，データ操作システム，汎用情報検索システム，情報管理システム，ファイル管理システムなどがある．初等的なシステムは，単純なレコード構造をもった順呼出しファイルを走査し，簡単な書式の報告書を作る．手のこんだシステムは，インデックスやリンクを用いていくつものファイルを扱い，オンライン方式で働く．システム間の方式の相違は，この報告書に示されている機能の種類と同じぐらいたくさんある．したがって利用者は，現在利用可能なシステムを評価選択するのに，困難を感ずるだろう．

システムズ委員会の調査では，最も大きい相違は，

親言語方式をとるか，独立言語方式をとるかである．親言語方式の例は，IDS や IMS である．独立言語方式の例は，GIS, MARK Ⅳ, NIPS/FFS, TDMS, そして UL/1 などである．

システムを親言語方式と独立言語方式とに，明白に二分するほうがよいという意見もあった．しかし，反対もあった．というのは親言語方式でも独立した機能体系をそなえるものや，独立言語方式でもふつうのプログラミング手法を用いるものがあらわれてきたからである．

この二つのクラスの相違を単純に明示することはむずかしい．委員会のだいたいの意見は，このようなシステムのクラス間の相違は，いずれは解消してゆくだろうということである．二つのクラス間では，プログラムの書き方に違いがある．この点はむしろ簡単に述べることができる．

親言語方式の場合には，これは応用プログラムの一つの新しい機能が追加されたものであるといってよい．つまり，COBOL, PL/I, アセンブラ言語などを親言語として，データベース操作の機能が埋め込まれるのである．

独立言語方式の場合には，これはプログラマばかりでなく，プログラマでない人のための道具でもある．手続き的な言語（つまり手続き向き言語）ととくに関係はなく，それ自体で独立して，閉じている．

それぞれのクラスの機能を詳しく説明するにあたって，多くのシステムのデータ構造は COBOL のそれよりも強力なものであることを指摘しておこう．委員会の意見によれば，このように強力なデータ構造は，現在の計算機応用にとって真に不可欠のものである．

強力なデータ構造の取扱い方は，二つのクラスでは異なっているし，とくに親言語方式の場合には変化に富んでいる．

親言語方式の場合には，COBOL ふうの階層構造をもっと大規模にしたものを用意しているか，あるいはレコード間の網目構造の表現を用意しているかである．後者はしばしば，階層構造をも含んでいる．

委員会が調べた範囲では，独立言語方式の場合にも COBOL ふうのデータ構造を強化したものを用意している．一般に階層構造はレコード内部に限定され，階層の個数は多重であり，可変長項目を認めている．

5.1 親言語方式

親言語方式というのは，COBOL, PL/I, アセンブラ言語などの手続き向き語語の機能のうえに作られたシステムのことである．前節で述べたような複雑なデータ構造を操作するために，利用者は，直接呼出しの低速記憶にあるデータベースと高速記憶とのあいだのデータ転送を要求する命令を出す．このとき，実際に物理的な転送が行なわれるかどうかを，利用者が知る必要がないようにシステムを作ってあるのがふつうである．

これらの機能と親言語とのあいだは，親言語中の CALL 命令によって連絡するのがふつうである．もっとも，より巧妙な連絡法を用いている例もある．一般に，親言語の性質とその翻訳ルーチンとは，変更しないですむようにしておくことが多い．

親言語方式の利用者は，いろいろな命令の列を実行される順序に合わせて書かなければならないので，要するにふつうの応用プログラマと同じことである．利用者は，ふつうの COBOL でプログラムを書いているときと同じ程度に，制御手続きの詳細を知っていなければならない．データベースシステムは，利用者の要求するデータ転送を，解釈して実行するだけである．プログラム中の論理的な流れは，条件文，実行文，ループなどを用いて，利用者が自分で組み立てる．もちろん，拡張されたデータ構造は親言語だけの場合よりも取扱いが容易である．媒体の種類とファイルのレベル構造は指定しなければならないが，記憶装置の物理的な構造は知らなくてすむ．

データ構造は，二段階に分けて記述することがある．たとえば，レコード内部のデータ構造は，COBOL のレベル構造の記述をそのまま用いる．それにたいして，レコード相互の構造関係は，データベース操作システムに固有の方式で記述する．多くのシステムでは，レコード相互の関係は変換されて実行時の内部テーブル（登録簿）となる．これについては，**5.4** で述べる．

5.2 独立言語方式

独立言語方式はいろいろなところで開発された．これはデータベースの機能の集合を，通常の手続きプログラミングによらないで取り扱うことをねらっている．各機能はかなり高水準のものであって，一般のプログラム言語の形式では，数多くのパラメータを必要とする．独立言語方式では，利用者の書く量を最少にするために，条件と動作との組合せはあらかじめ固定してしまう．このような方式はしばしば非手続き的とよばれる．すなわち，利用者は論理的な制御を詳細に順をおって書くことはしない．とくに，データの検査，転送，転記の順序を制御することはできない．

独立言語方式においては，データ構造の記述は符号化されて，データのカタログまたはファイル中に，いっしょに保持される．

調べたかぎりでは，独立言語方式で最も広く採用されているのは，問合せと更新の機能である．ここで問合せというとき，データの選択，分類，報告書作成を含める．報告書作成の機能はさらに一般化されている場合もある．問合せ機能を有するシステムは汎用照会言語あるいは汎用情報検索システムとよばれる．

更新の機能はこれほどまでには一般化されていないが，拡張されたデータ構造を有する独立言語方式のシステムにおいては，アセンブラ語でめんどうな更新手続きを書く手間を避けるために，更新機能が組み込まれているのがふつうである．

問合せや更新の機能を用いるときにデータ記述を入力する必要はない．これはあらかじめデータそのものに付されていてシステムがこれを解釈するのである．

独立言語方式には，ファイルの生成と再構成の機能も含まれる．ファイルの生成にあたっては利用者の定義したデータ構造の記述を，上述の符号化された形式に変換することも行なわれる．また，ファイルに含めるデータの正当性確認も行なう．ファイルの生成はファイルの第1回目の参照に相当するが，これはたいていは，空ファイルにたいする更新機能として実行される．

再構成機能はあまり一般化されていない．データ構造の記述を変更しファイルとレコードとを写像する．

現在の独立言語方式のシステムの弱点は，適用できる応用の範囲が限定されていることである．ただ，この範囲内では新規の問合せや更新にあたっての，プログラミングなどの準備の時間を大幅に減らすことができ，臨時の要求にたいしても手早く答えることができる．

独立言語方式における実行時の機械時間は，もちろんその作りにもよるが，一般にプログラミング方式の場合よりも余計にかかるようである．

5.3 データの互換性

現在の利用者にとって最大の問題点は，データベース操作の二つの方式に互換性がなく，さらに既存の手続き言語とのあいだにデータの互換性さえもが保証されないことであろう．これは主として，既存の手続き言語よりも複雑なデータ構造を有するためである．一つのクラスのなかでも，データ構造の拡張の程度がいちじるしいので，互換性が保てない．しかし，既存のプログラム言語で扱えるファイル構造に限定した独立言語方式のシステムもある．

データの互換性のトラブルはしだいに大きくなっているし，データベース操作に二つの異なった方式があることがそれをさらに悪化させている．データの互換性は，記憶のいろんなレベルで問題になる．一つの直接呼出し装置中の複数個のファイルをどうやって識別するか，ファイル中のブロックの配置，ブロック中のレコードの配置，レコード中の項目の設計などを考えなければならない．一次索引，二次索引，レコード間のポインタなどの拡張機能は，データの互換性をさらに困難にする．実際，COBOL においても，プログラムの変換よりもデータファイルの変換のほうがずっとむずかしいことが知られている．

この，データ交換の問題は，第10回 CODASYL 年会（1969年5月）においても，かなり討論され，これを解決するためのデータ記述言語が提案された．システムズ委員会では，記憶構造記述言語を研究する作業委員会を設けた．

5.4 データ独立および結合

データベース操作システムは，データ独立でなければならないとしばしば考えられている．データ独立という用語は厳密な定義はないが，データとそのプログラムとのあいだが分離していることを示すものである．その分離の程度にもよるが，データの記述や構造を変更しても，同じプログラムがそのまま修正や再コンパイルなしに使えることが期待されている．

データ独立にするためには，データ構造の記述を実行時に保持していなければならない．実行時に保持しているデータ記述を解釈して，ファイル中のレコードやレコード中の項目の位置ぎめを行なうことを，実行時結合という．通常のプログラミング方式では，データとプログラムとの結合は，翻訳時に行なわれる．

ある種の親言語方式では，混合型の結合が用いられている．すなわち，項目への結合は翻訳時，レコードへの結合は実行時に行なわれる．この場合，レコード相互の関係の記述を表（登録簿）に入れ，これを実行時に参照できるものとしておかなければならない．

独立言語方式は，データ構造の記述をこのんで実行時に保持する．結合のための実行時間は，データ構造の記述がオブジェクトコードに変換されていたりするので，必ずしも長くかかるとはかぎらない．この変換は問合せまたは更新の命令が出たときに行なわれる．

理論的には，結合をどの方式で行なうかということ

と，親言語と独立言語というクラスとは，どうにでも組み合わされる．現在ある親言語方式のシステムは，混合型の結合を採用しているが，将来ともそうでなければならない理由はない．

5.5 利用者との関係

データベース操作システムとその利用者との関係，とくに利用者の学習しなければならない事項について述べる．

親言語方式では，応用プログラマは親言語にたいする拡張点あるいは CALL 命令のパラメータを学ばなければならない．

独立言語方式では，まったく新規の言語を用いる．プログラマでない人にとっては，これが従来の手続き的言語と異なっているということは，トラブルにはならない．また，COBOL を知っている人にとっても，独立言語方式のシステムを利用するほうがCOBOLでプログラムを書くよりも楽であろう．システムのなかにファイル処理の手続きが組み込まれていて，それをいちいち自分で書きおろさなくてもすむからである．

現在の独立言語方式のシステムは，命令の形式をCOBOL ふうの書き方に近づける努力はまったくはらっていない．

5.6 利用者の水準

利用者の水準として，データ管理者，応用プログラマ，プログラマでない人，パラメータ利用者の四者を区別する．

どのシステムにおいても，データ管理者という職務が必要である．オンラインの利用者が大勢いるときには，重要なデータベースの管理責任者がいなければならない．管理者は，データベースの最初の生成と，その構造の変更に責任をもつ．汎用データベース操作システムの諸利用のうち，利用者の資格を限定しなければならないものについて，システムズ委員会は調査した．データ管理者専用の機能を利用するさいに必要とされる専門的技能の程度は，システムによっていちじるしく異なっている．

次の水準の利用者は，親言語の知識のある応用プログラマである．その職責は，現在よく知られているとおりであるが，オンラインのデータベースを扱うプログラムには，従来の事務応用のプログラムにはなかったような，特有の制約がある．

3番目の利用者は，プログラマでない人である．この人たちは，独立言語方式を用いるものと想定されてきた．

最後は，パラメータ利用者であって，これはプログラマでない人のうちの一部である．データベースを利用するときには，あらかじめ定義されているトランザクションを呼び出し，そのパラメータに値を与えるだけである．

6. 設計者の技術的問題点

汎用データベース操作システムの設計方針にはいろいろある．そのために，新規の設計者はそのいずれを選ぶかに，迷ってしまうだろう．ソフトウェア開発へなされた大きな投資と，計算機科学の急速な進展は，不幸なことに，他社の誤りを認識しそれから学ぶことをたいへん困難にしている．

システムズ委員会は，現存の汎用データベース操作システムの調査研究に3年間をかけ，その将来の方向を明示しようとした．この報告書は将来の方向を提示するためのものではないが，いくつかの示唆は含めてある．

6.1 現在の記憶構造

現在すでに，データの蓄積にたいしてばく大な投資がされている．これらのデータをすべて処理できるようなシステムを開発することは，あきらかに不可能である．しかし，今後のシステムは，ふつうの記憶構造は扱えるようにしておくべきである．そのために，実行時にデータを解釈する方式が可能であろう．

翻訳時に結合を行なうシステムが始終利用しているデータを，ときにこちらから利用したいような場合にこの方式をとれよう．しかし，将来はこちらのシステムのほうが重要になるのならば，データを変換してしまうほうがよい．

広く使われているふつうの記憶構造を，どれでも扱えるようにしておくことは，今後の設計者の留意すべき事項である．

6.2 複雑なデータ構造

ある種の応用では，現在の COBOL におけるレコード内部の構造やレコード間の構造よりも，もっと複雑なデータ構造が要求される．CODASYL プログラミング言語委員会データベース作業班 (DBTG) の1969年10月報告書[3]では，レコードを網目構造にして格納できるファイルを提案している．DBTG の提案は二つの部分からなる．第一は，DDL (データ定義言語) であって，データの構造などを記述する．第二は，DML (データ操作言語) であって，構造化されたファイル中のレコードにたいする動作を記述する．

DBTG は，データ定義言語とデータ操作言語を分離することが必要だと考えている．分離によって，データ定義言語で記述されたデータベースと，それを処理するプログラム言語とを独立にできる．

DBTG では，そのデータ記述言語は FORTRAN や PL/I などからも利用でき，また，独立言語システムから利用されることもありうると期待している．もっとも，DBTG としては，データ操作言語として COBOL を拡張したものを提案している．

システムズ委員会の報告書では，この DBTG の提案を親言語方式のうちに含めて検討した．

6.3 独立言語方式による網目構造の処理

独立言語方式のシステムは一般に，データの階層構造だけを扱い，網目構造は扱わないことが目立った．その理由は，階層構造のほうがより広く理解され，利用されていることによるのであろう．実際の応用において網目構造を利用しなければならないものが何%くらいあるかについては，議論の余地があるが，網目構造を導入する必要があることは確実である．網目構造の利点が認識されれば，さらに広範囲に利用されるようになるにちがいない．

この技術が順呼出しファイルの技術と同じくらい普及すれば，独立言語方式にあるようなまとまった機能が定義できるだろう．

6.4 プログラマでない人による利用

設計上の問題は，大きな共用のデータベースをプログラマでない人に利用させたいという要求からもくる．この利用は，オンライン端末（タイムシェアリングというあいまいな用語は避けるべきである）によるのがふつうであろうが，一括処理によってもよい．

まず，パラメータ利用者についてみよう．利用者は既存の手続きを呼び出し，パラメータの値を与える．この手順をトランザクション，既存の手続きをトランザクションプログラム，パラメータの値をトランザクション入力とよぶ．プログラムの機能は，問合せおよび更新である．銀行業務や会計では，トランザクションという用語は，更新に関してだけ用いられる．

パラメータ利用者は，トランザクションプログラムの作成やその言語については何も知らない．知っているのは，トランザクションの種類とその出力の意味だけである．

親言語方式のシステムのいくつかは，オンラインのパラメータ利用者を扱うことができる．トランザクションの件数が多い場合には，そのための特別の機構が必要になる．座席予約などの特定業務では，こういう機構は広く組み込まれている．件数が多い場合には，待ち行列やチャネル使用の優先権を定めることが多い．この分野の問題は，しばしばデータベースの問題の一部とされてはいるが，実際にはそれ自身で一つの独立したテーマとなるものであり，システムズ委員会では，綿密には調査しなかった．

トランザクションプログラムを作成するのに，通常は，なにかの親言語を拡張したものを用いている．独立言語方式のシステムで，大量のトランザクションを扱う技術に大きく貢献したものがなかったことを指摘しておこう．

プログラマでなく，パラメータ利用者でもない人は，自分の問合せや更新を定式化するために，その言語を学習しなければならない．システムと対話を重ねることによって，問合せの文法を修正して正しいものにし，求める効果を得る．

独立言語方式のシステムは，対話方式をとっているので，これらの人たちにとって，より魅力的に思われる．

6.5 二つの方法の統合

パラメータ利用者やプログラマでない人のために統合されたシステムを開発することは，汎用データベース操作システムの設計者たちの主要な問題である．一つの記憶構造が，どの利用者にも使えるようにしておかなければならない．前もって定義されたトランザクションプログラムは，親言語方式のほうにあり，対話方式は，独立言語方式のほうにある．この二つの方法を統合することが可能であると委員の多くは感じた．すなわち，トランザクションプログラム中に適当な独立言語方式の機能を入れ，また，トランザクションプログラム中で必要な機能を親言語で追加できるようにしておくのである．

一つのデータベースが，手続き向き言語からパラメータ的な言語までのどの水準の言語によっても生成でき更新できることが重要である．さらに，データ定義の過程とそのデータを操作する言語とは，分離しなければならない．データ定義は多くの言語に共通でなければならない．

7. 開発の基礎としての COBOL

CODASYL のどの委員会も，データ処理社会における COBOL の役割に関心を抱いてきた．COBOL が 10 年ばかりのあいだに世界で最も広く使われるプ

ログラム言語になったということはだれも否定できない．FORTRAN と比較した場合，このことは計算機が科学用よりも商業用のほうに広く使われていることを反映している．PL/I と比較した場合，このことはCOBOL のほうが数年はやく世に出たことを反映している．他方，COBOL は多くの論点から批判されてきた．機能をつめこみすぎているといわれる．また，その冗長さが批判され，さらに PL/I や ALGOL のスマートさが強調されている．

機械と通信するために人間が使うプログラム言語は人が仲間と通信するために使う自然語と同じくらいに，一片の法律だけで取りかえることはむずかしい．英語は，不合理なつづりやあいまいさのゆえに広く批判されてきたが，西欧世界では最も広く使われている言語である．新しい諸要求を満足するために COBOL を改良することはできるだろうが，うまく一挙に取りかえることはできそうにもない．CODASYL のプログラミング言語委員会は，COBOL の拡張を担当し，毎年，何百という提案を検討している．

7.1 データ構造

共通の汎用データベース操作システム開発のためにどこまで COBOL を使っていけるかは議論のある点である．多くの人は，COBOL のデータ構造は，一括処理には適しているが，データベース用には制限が強すぎると感じている．**6.2** で述べたように DBTG は，網目構造を定義できるデータ構造を提案した．そのような構造は，事務データ処理におけるレコード間の複雑な関係をより直接的に表現し，単一順序の階層構造ではできないような多くの異なった順序づけを表現できる．

多くの汎用データベース操作システムの開発経験によって，DBTG の提案以外にレコードや項目の段階でのデータ構造の拡張も必要であることが示された．たとえば多くのシステムは，長いストリングデータの可変長項目を処理するために，レコード内部の複雑な構造を許す．この拡張はまた，文字単位のストリング処理の機能をも要求する．

DBTG の提案したレコード間の関係の記述と，他のシステムにおけるレコード内部の構造の記述とは，相互に排他的なものではないことを指摘しておこう．ただここで，レコード間の構造の記述を複雑にすれば，レコード自体はより小さくより簡単なものとなり，かつ論理的な呼出しを行なう傾向に通じる．またレコード内部の構造の記述を複雑にすれば，レコードはより大きくより複雑なものとなり，かつ大きなデータ単位をより少なく呼び出す傾向に通じる．

7.2 親言語方式

汎用データベースシステムの親言語として COBOL を採用し，**7.1** で述べたようなデータ構造を定義記述し，その共用データを操作する機能を拡張することができる．DBTG によるデータ操作言語 (DML) の提案は，手続き部のそのような拡張である．

データベースの構造と応用プログラムの構造とのあいだの写像を利用者が自分で定義することにすれば，COBOL その他の既存の言語を拡張しないでも，データベース応用のプログラムを書くことができる．応用プログラマは自分のプログラムに適したデータ構造を記述することができる．このやりかたによるならば，異なったプログラム言語のあいだで，矛盾のない拡張を行なうことができる．DBTG のサブスキーマの概念は，このやりかたで写像を用意するものである．この概念の応用は，今後の課題である．

7.3 独立言語方式

独立言語方式については，企業間での共同開発は行なわれていない．COBOL の開発と DBTG の提案は，プログラマ向きの機能である．以前は，同じデータベースが親言語にも接続できるようにすることが重要だとされた．DBTG は，データを処理する言語とは独立にデータを記述するデータ定義言語 (DDL) を提案した．データ定義を外部に設定することによって，親言語も独立言語も，現在あるものも新しく開発されるものも，同じデータベースを処理できる．この目的に適した独立言語の仕様は，研究課題として残っている．

COBOL を基礎とした場合，問合せや更新などの独立言語に典型的な機能は，新しいディビジョンの形で達成できるだろう．これらのディビジョンは，データ管理者があらかじめデータ部を用いて定義しておいたデータを処理する．内部表現にしたデータ定義はデータといっしょに格納してもよいし，そうしなくてもよい．その程度は，独立言語方式の例に見られるとおり，さまざまでありうる．

DBTG の DDL では，外部のデータ定義に名前をつけて参照する機能を提案している．上でも述べたように，同じデータベースを COBOL 以外の言語や非手続き的機能で処理することも可能であろう．

8. 汎用データベースシステムの機能

この章は，汎用データベース操作システムの諸機能

を 10 節に分けて述べる.

8.1 要 約

ここでは,各システムにおける諸機能を概括する.

8.2 データ構造

ここでは,システムの利用者からみたデータ構造を述べる.記憶技法の詳細は以下の各節で述べる.各種のデータベースシステムのデータ構造の理解は,その能力の十分な理解には欠くことのできないものである.前述のように,多くのシステムは多少とも COBOL とは異なったデータ構造を用意している.

システムズ委員会は,すでに 1969 年5月の報告書において,データの論理的な構造と,記憶の物理的な構造(秘密にされることもある)とを区別することの重要さを確認した.

多くのシステムは,階層構造と網目構造を扱う.

データ構造の階層は,項目,集団,集団関係,レコード,ファイル,そしてデータベースとする.データ構造の定義は,報告書全体を通じてスキーマとよぶ.いくつかのシステムは,スキーマの部分集合としてサブスキーマを有する.

8.3 データ定義

データ定義はデータ構造と密接に結びついている.この節ではデータ構造を扱うスキーマおよびそれを記述する言語や表を,前節のデータ構造の階層にしたがって述べる.さらに,データ定義の入力法および **5.4** で触れられた結合という重要概念をも論ずる.

8.4 問合せ

問合せの機能によって,データベース中のある部分を選び,抜き出して,それを印刷などの形で表示する.前半はその部分の選出であり,後半は計算,分類,書式作りなどである.問合せはそれ自身で独立した機能であって,利用者は質問を組み立てるだけでよく,データベースを呼び出し,情報を抜き出す手順を詳細にコーディングする必要はない.

問合せを行なうためには,その処理手順がシステムに組み込まれてなければならない.最も簡単なものでは,格納されているファイルを順に探索し,所要のレコードを写し出し,レコード中のデータを組み立てて報告書を作る.基本的な順次探索の手順ですら,複雑さの程度はさまざまであるが,順次探索以外にもファイルを呼び出して探索する手法はいろいろある.

8.5 更 新

更新の機能はデータベースのある部分の内容の値を変える.「更新」というときには,格納してあるデータ定義を修正するようなデータ構造の再構成は除く.更新にあたって対象となる部分を最初に選ぶのに,問合せと同じ手順を用いる.独立言語の多くのシステムでは,選択機能は問合せ機能中のものと共通にしてある.そして,選択された部分は表示されるのではなく,ある指定にしたがって変更されるのである.

更新は,それ自身で独立した機能であり,組込み手順とできる.その作成手法は,問合せの場合よりももっと変化がある.いくつかのシステムでは,データベース中のファイルを1回通読するあいだに,更新と問合せの両方を行なう.

8.6 生 成

生成にあたっては,データ定義を準備し,ファイルを作るためのレコードを用意しなければならない.そのほかの機能として,データの正当性確認,機密保護の仕様,記憶構造の制御がある.データベースの生成は,データベース管理者の重要な職務の一つである.ファイルの生成は,問合せや更新と同様に組込み手順にしてあることもあるし,また通常のやりかたでプログラムしなければならないこともある.多くの場合これは,空ファイルにたいする更新機能の適用である.

ファイル生成においては,独立言語方式と親言語方式とのあいだに,大きな違いはない.いくつかの独立言語システムでは,ファイル生成の機能が組み込んでないために,最初のファイルはシステムとは別にプログラムを書いて作らなければならない.

8.7 プログラマ機能

プログラマ機能は,親言語方式中にある.これは,利用者が親言語でプログラムを書くときに呼び出せる機能である.最も重要なのは,データベースと高速記憶とのあいだの転送の命令である.そのほかに,開く,閉じる,待ち合わせるなどのファイル管理の命令もある.

プログラム言語の機能であっても,データ管理者の職務に属するものは,この節ではとりあげない.また独立言語システムから手続き向き言語へリンクする機能(オウンコード)もとりあげない.

8.8 データ管理者機能

データ管理者は,データベースの責任者である.親言語システムでも独立言語システムでも,その役割は認識されている.データ定義とファイル生成の重要機能は,すでに述べた.データ管理者は,システム動作の監視,システムの保全と機密保持,新しい種類のレコードや項目を収容するための再構成などをも行な

う．システムによっては，データ管理者機能に，プログラム言語を用いなければならないものもある．このような言語機能のうちのどの範囲をデータ管理者機能とするかは，主観的なものとなろう．

8.9 記憶構造

データ構造の各階層の内部表現を記憶構造という．ファイルの記憶構造は，レコードが物理ブロック中にどのように記憶されてファイルの内部表現を作るかを定義する．これは，データ管理あるいは入出力制御とよばれるシステムによって管理される．記憶構造としては，そのほかに，索引順次ファイルなどのファイルおよびレコードの編成，呼出しの手法もある．

レコードの記憶構造は，システムによって大きく異なる．それは，集団あるいは項目が記憶装置中でどのように表現されて，レコードの内部表現を形成しているかを定義する．しばしばレコードは，低速記憶中で隣接して記憶されるが，またいくつかのシステムでは，集団がセグメントに写像され，一つのレコード中のセグメントが，低速記憶中のばらばらの場所に格納される．項目の記憶構造は，システムによってその写像の制御方式にいろいろあるが，とにかくある程度はハードウェアの記憶装置の特性を反映している．

8.10 操作上の特性

現在あるシステムは，ある種の操作環境のもとで働くように作られている．この環境は，ハードウェアの構成とオペレーティングシステムによるソフトウェアの環境とからなる．この節では，そのシステムの機能のうち，ほかのソフトウェアと関係するインタフェースの部分だけを述べる．

9. 将来の開発におけるシステムズ委員会の役割

システムズ委員会は，汎用データベース操作システムの調査に3年間を費やした．この調査がそのように長くかかったということは，この問題の複雑さがますます増大していることを示す．多くの組織がそのような長期間にわたって委員を参加させ，活動を援助したのは，関係者にとってこれらのシステムの理解が重要であったためである．

システムズ委員会が二つの調査を出版する理由は，その知識をデータ処理社会に提供するためである．委員たちは，汎用データベース操作システムのもつべき機能について重要な展望指針を与えたと考えている．二つの報告書の目的の一部は，この展望を普及し，このようなシステムを考え理解するにあたっての一般的な規準を立てることであった．CODASYL システムズ委員会の規約は次のようなものである．

「システムズ委員会は，システム解析，設計，作成などとよばれている手順をできるだけ自動化するという目的をもって，データ処理における，より進んだ言語と技術とを開発するに必要な専門的知識を確立するよう努力する．」

この報告書によって，データベースについての専門的知識は確立されたといえよう．次の段階は，共通システムの仕様である．システムズ委員会は，この報告書中の知識と展望とが，今後のシステム開発にあたって利用されることを期待している．

プログラム言語の開発を委員会組織で行なうことは，従来はあまりうまくいかなかった．希望的な材料としては，グループが共同で作業し，互いに討論し，また外部とも連絡を保つことを，現在までに学んだということである．開発作業のように野心的な活動を試みるときには，このことはあきらかに重要なことである．

（訳　西村恕彦・星　守）

参考文献

1) CODASYL Systems Committee: A survey of generalized data base management systems†. May 1969. Available from ACM at $7.00 prepaid.
2) CODASYL Systems Committee: Feature analysis of generalized data base management systems††. April 1971. Available from ACM, New York City, British Computer Society, London and IFIP Administrative Data Processing Group, Amsterdam.
3) CODASYL Data Base Task Group: October 1969 Report†††. Available from ACM New York City and IFIP Administrative Data Processing Group Amsterdam at $4.00 prepaid.
4) CODASYL COBOL Journal of Development 1969††††. Number 110-GP-1a. April 1970. Available from Canadian Government Specifications Board, Ottawa, Canada.

† 小林功武編著：データ・ベースの管理，企画センターに訳出されている．
†† この序論の本体．ACM より $8.50 で発行．
††† 1971年4月に改訂版が出て，プログラミング言語委員会はその大筋を承認した．ACM より近刊予定．
†††† 1970年版が発行された．番号は 110-GP-1b．

（昭和46年5月24日受付）