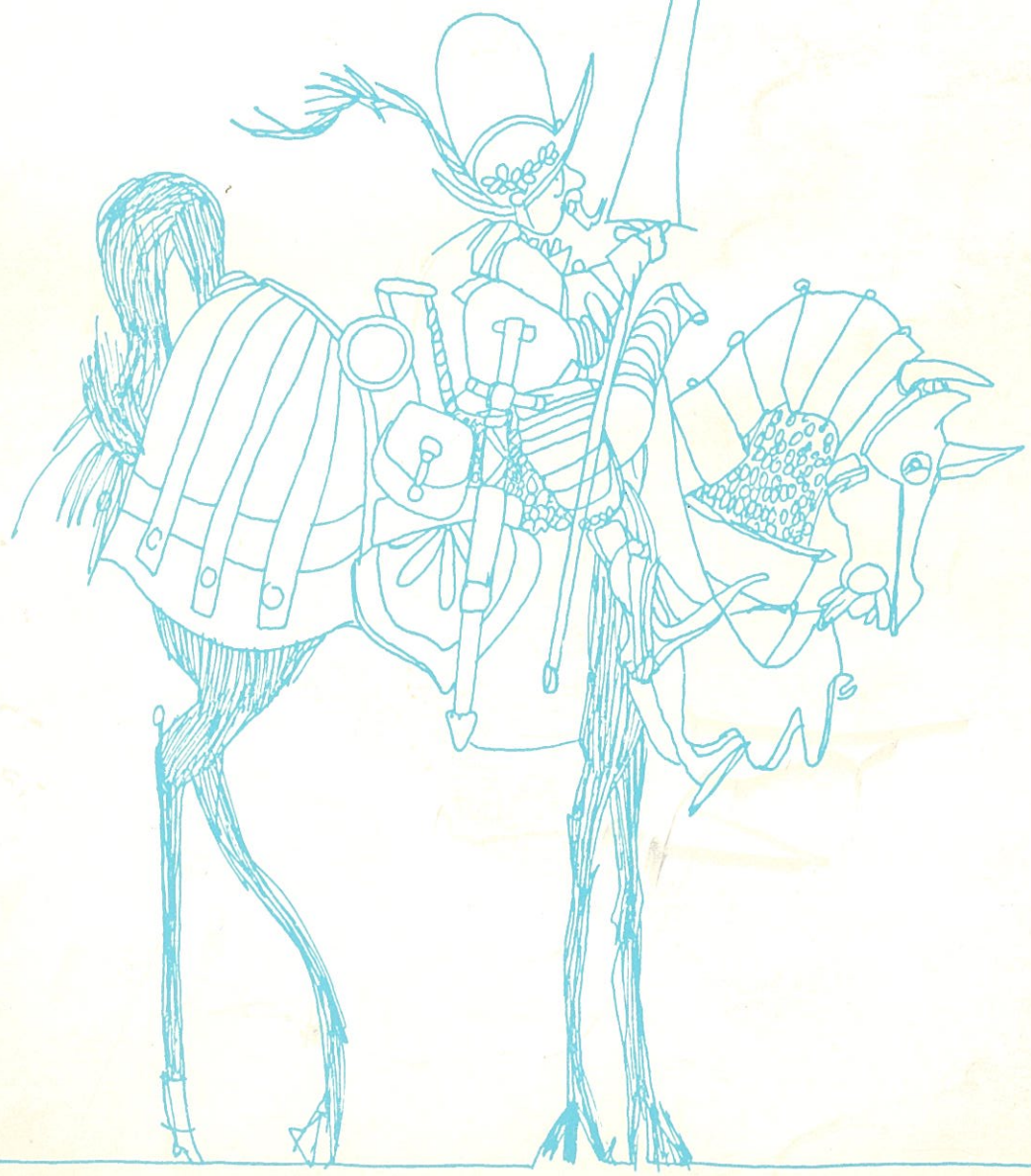
TOTAL

データ・ベース
マネジメント
システム

概念と諸機能



目　次

まえがき

第1章　TOTAL の概念

第2章　TOTAL 概　説

第3章　データ・ベース構造

第4章　データ・ベースの定義

第5章　データ・ベースのフォーマッティング

第6章　データ・ベースへのアクセス

第7章　TOTALのオペレーション・モード

第8章　データ・ベースの保護

第9章　データ・ベースの変化

第10章　ま　と　め

TOTAL

データ・ベース・マネジメント・システム

概念と諸機能

このマニュアルはTOTALデータ・ベース・マネジメント・システムについて，その概念と諸機能を紹介します。読者はマニュアル「なぜ　データ・ベースが　必要か？」を読まれるか，㈱アシストのセミナー「なぜ　データ・ベースが必要か？」に出席されておくことが望ましい。

第3版　昭和50年8月

第1章　TOTALの概念

この章ではTOTALの意義について説明し，次のような疑問に答えようとしている。

＊　TOTALは何を目的とし，開発されたものか？

＊　TOTALはあなたの企業にどのような利益や利点をもたらすことができるのか？

なぜ　データ・ベース・マネジメント・システムが必要か

データは，今日のどの企業においても，最も重要な資産の1つである。

会社では，何千，何万人という社員が，毎日，業務や意志決定を行っているが，そのうちのごくわずかな人だけが会社の得意先に会い，会社の購入部品，棚卸商品在庫を管理し，会社の資金を扱っているのである。他の人達，つまり管理者や一般社員の大多数は，会社の得意先，購入部品，棚卸在庫，資金等といったものについてのデータを基に意志決定を行い，それぞれの仕事を遂行しているのである。

従って，今日の企業，特にコンピュータによる処理に年間総経費の1％またはそれ以上の費用をかけている企業では，各層の管理者や一般社員に対して，必要な人に，必要な時に，必要な形で直接データを使うことのできるような，コンピュータをベースとしたシステムをつくり上げている，と考えることは理にかなっているのである。しかし，このことがどんなに筋の通っていることであっても，もちろん事実は理論からほど遠いのである。今日のほとんどの会社では，一部の専門家や技術者グループの助けなしで，コンピュータをベースとしたデータを手に入れることはできない。これら専門家グループの数は不足しているし，多忙であって，しばしば過度に労働を強いることにもなるので，それぞれの人の要求に合ったピッタリした情報を提供することはごくまれにしかできないのである。

その結果，情報を必要とする管理者や社員は，定期的に出される週報とか月報に頼ることになる。これらのレポートは，どのユーザーの希望にも完全に応ずることができるわけではないが，その代わりに非常に多くの「情報を利用する人」がある程度まで満足するようにつくられているのが普通である。管理者や事務の担当者が，必要としているレポートの作成日が来る以前に情報を必要とする場合には，ソロバンに頼ることになる。また，定期的に作られるレポートでは得られないような情報を必要とする時には，「ソロバン」を使って計算するか，専門家が新しいレポートをつくるまで，数日，数週間，長いときには数ケ月も待つ必要が生じる。通常の場合ソロバンの方が早いことになってしまう。

我々が行なってきた情報システムの機械化において，最も重大な結果の1つは，今日の企業内でのデータが，大多数の管理者や社員にはほとんどアクセスできないように設計されてきているということであって，これは確かなように思う。

非常に多くの会社では，ここ数年間，データ・ベース化を指向してきているようである。この理由は，それぞれの仕事を行う上で情報を必要としている会社の人達に対して，データをよりアクセスが簡単にできるよう，より有効に活用できるようにしようとの要望に応じようと考えているからである。

このような要望にこたえられるかどうかは，データ・ベースを処理するために使われるソフトウェアに大きく依存しているのである。このようなソフトウェアは，データ・ベース・マネジメント・システムまたはDBMSと呼ばれている。CODASYL，GUIDE & SHAREといったようなグループでは，この要望にこたえるために，DBMSを満足させる基準を定めるべく広汎な検討を重ねてきている。この基準は，TOTALをデザインする際に使ったものと大部分が同じものである。この基準について次に考えてみよう。

TOTALの設計基準

利用上の容易さ

DBMSは情報を利用する人がデータをアクセスする場合に，コンピュータの専門家に依存しなければならない程度を増やすのではなく，むしろ非常に少なくするはずのものである。もしもDBMSが非常に難しいもので，コンピュータの

専門家でさえ，数週間または数ケ月間のかなり専門的な訓練を受けなければ使うことができないのであれば，情報は使いたいが自分自身コンピュータやDBMSの専門家ではない一般管理者や社員には，データを一層アクセスし難いものにしてしまうだけである。

この点で，TOTALが非常に利用し易いものであることを，このマニュアルを通じて十分知っていただけるものと期待している。どの企業にとっても，TOTALが貢献できる最も重要なことの1つは，データをより直接的にかつ容易にアクセスできることであると確信をもって申し上げられる。

モジュール構造と発展性

データが今日の企業における重要な資産であるということを認めるときに，考えなければならない別の重要なファクターは変化ということである。我々は急速に，しかもその速度を速めながら変化している時代に生活している。最近出版された，アルビン・トフラーの「未来の衝撃」と題する本に書いてある次のような事柄を考えてみたい。

＊ 人類が生存するようになって以来約50,000年をおおよそ各62年のライフタイムとして分けた場合には，約800回のライフタイムがあるという計算になる。

＊ この800回のライフタイム中で，約650回はほら穴ですごした。

＊ 次の75回のライフタイムで，初めてライフタイムからライフタイムへの効果的な伝達が可能となった。

＊ 過去6回のライフタイムで，初めて人類の大多数が印刷された文字を読めるようになった。

＊ 最近4回のライフタイムで，初めて正確に時間をはかることができるようになった。

＊ 最近2回のライフタイムで，初めてだれでもが，どこででも電気モーターを使えるようになった。

＊ 今日我々が日常生活に使っているすべての必需品で圧倒的大多数のものは，現時点，第800回目のライフタイムで開発されたものである。

＊ これまでに生まれた全科学者の90％までが現在生存しているのである。

＊ 今日消費者が購入する製品の55％が10年前には，存在していなかった。

もちろん，企業はこのような変化から離れているわけにはいかない。事実，企業は我々の社会におけるどのような制度よりも，より早く，おそらく最も大きく変化するのである。このことは新製品とか，新しいマーケットとか，仕事の計画，および実行のための新方式等についていえることである。

仕事が変化していく場合には，その仕事についてのデータも，そして会社の持っている情報への色々な要求も，それに従って変化するのである。このようなことから，コンピュータへの投資に重点を置いている会社では，変化に適応できるようなコンピュータをベースとした，拡張性のあるシステムをつくりあげていると考えられるのであるが，これは本当であろうか？　明らかにこれは正しくないのである。今日までに我々がつくり上げてきたコンピュータをベースとしたシステムや，今も作りつつあるシステムは，非常に融通のきかないものである。つまり，今日の企業において，EDPに費やしている設備や人やお金の半分以上が既存のシステムを変化に適応させるために使われているのである。

このような膨大な，事実上法外な浪費は不要なことであって避けることができるのである。

TOTALデータ・ベースは完全にオープン・エンドなモジュール構造になっており，拡張が容易なものであることをこのマニュアルを通じて皆様に知っていただきたいと思う。TOTALデータ・ベースは，仕事に関する情報要求が増えたり，変化したりする時には，それに応じて増やしたり変化に適応させていくことができる。大部分の変化というものは予測できないものであり，だから変化を織込んだ計画などできるはずがないということは事実である。しかしTOTALは"変化"を気にしないで，データ・ベースをデザインすることができるのである。図1－1参照。

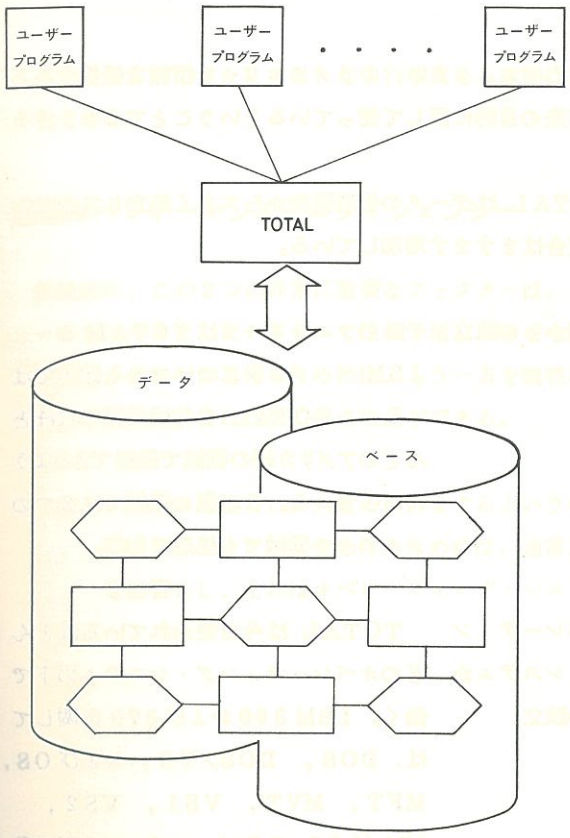


図１－１ TOTALの位置

データの独立性

次のような事実を考えてみたい。

1. データ処理に要するコストのうち人件費の割合は非常に大きくなってきた。
2. 50％以上のプログラミング時間と費用とが，既に出来上がっているプログラムをメインテナンスしたり変更したりするのに使われる。
3. 大部分のプログラムのメインテナンスを行なう理由は，そのプログラムが処理するファイルと密着しているからである。

もしも，プログラムを，それが処理するデータから独立させておく方法が見つかるなら，プログラミングをするための努力，すなわち，EDPにかかる経費，のかなりの部分を実質的に減らすことができるのである。

TOTALには次のような方法が用意されている。

TOTALでは，アプリケーション・プログラムはその中で必要となるデータの項目にのみ関連をもつが，レコードとかセグメントには関連を持たない。従って，レコードとかファイルに実際に変化が生じたり，レコードやファイルがもつ相互関係が変化しても，アプリケーション・プログラムを作り直したりコンパイルし直したりする必要がなくなる。

データの相互関係づけによる重複データの排除

データ処理費用は高くなっており，それがますます急激に高騰している別の大きな理由として，多くの会社では膨大な量のデータがあり，それらを重複処理しているということが考えられる。

これは一体なぜだろうか？

この理由はデータそれ自体（すなわち，その内容）は，意志決定を行ない行動を起し監視するための基礎になっていないということである。むしろデータを理解することが，意志決定と行動との基礎になっているのである。そして，データを理解するためには，それを別なデータと関連づけてみる必要がある。

しかし，これまで伝統的に行なわれてきたテープ・ベースによるシークェンシャルなバッチ・データ処理システムは，1つのファイルに含まれるデータを別のファイルにあるデータと関連づけることが非常にできにくかった。結果的には，これまでの伝統的なデータ処理方法ではたくさんの異なるファイルに同一データをもち，多種類のレポート上に同一のデータをプリントすることであった。これらのファイルやレポートを使って，同一のデータをいろいろな観点から（すなわちデータの組合せをいろいろと変えて）見ることができるようにした。

その結果

* 同一のデータを複数のファイルに持ったために，テープやDASDのスペースを多く必要としている。
* データを維持するため（すなわち，複数ファイル内の同一データ間の矛盾をなくするため）および同一データを異なる観点から見れるような諸レポートをつくるために，多くの処理時間と費用とが必要となっている。

TOTAL はデータを格納している物理的ファイルには関係なしに，どのデータ項目でも別なデータ項目に関連づけるようにしたネット・ワーク構造で，データを組み立てることにより，重複データを排除している。

TOTAL を使って，会社業務の流れにそったデータ・ベースを作りあげてゆくことができる。TOTAL のデータ・ベースは，データのアクセスや更新が直接に，ごく自然に行なえるので，それだけ使い易いのである。

データの保全と保護

企業が所有している貴重な財産と同様に，データも次のことに対して保護される必要がある。

* 予期しない災害や事故による損害
* 権限を持たない人によるアクセス

TOTAL ではアプリケーション・プログラムがファイルやレコードを壊さないように保護しており，またデータの項目，レコードおよびファイルが認められた人によってのみアクセスされ，更新されるという，アクセスを制限する機能をもっている。TOTAL は，また，データ・ベースが壊れたときのために，データ・ベース更新のすべてのログとデータベースの復旧の優れた機能がある。

機械，オペレーティング・システム，プログラム言語などからの独立性

これまでのデータ処理での大きなむだは，変換に費やした人員，時間および費用であった。今日コンピュータを利用しているほとんどの大企業では，計算機およびオペレーティング・システムの変更に伴ういくつかの大規模な変換を何回か経験している。多くの会社ではバッチ処理からオンライン処理に至るまで，多くのアプリケーションを変換しているし，今後もこのような変換を行うことが続けられるであろう。

これまでの各変換では，既存のファイルとかプログラムを大きくデザインし直すとかプログラミングし直すといったことが必要であった。それぞれの変換に，多くの人手と途方もない費用を必要としてきたのである。そして，もっと重要なことは，この変換を行なうことによって会社は EDP 諸資源を会社内で，より良い，よりタイムリーな情報を提供するという本来の目的に反して使っているということであろう。

TOTAL はデータの処理環境から大きく独立しており，その度合はますます増加している。

機械からの独立　このマニュアルでは TOTAL の IBM バージョンについてのみ述べてはいるが，TOTAL 自体は最近ではほとんどのアメリカ製の機械で処理できるようになっている。日本製の機械では２つのメーカーの機械でも処理できる。

オペレーティング・システムからの独立　TOTAL は今日使われているほとんどのオペレーティング・システムの下で働く。IBM 360 および 370 に関しては，DOS，DOS／VS，および OS，MFT，MVT，VS1，VS2，VM／370 の各バァージョンの下で動くと同時に，GRASP，POWER，HASP，ASP などと一緒に使うことが可能である。

バッチまたはオンラインからの独立　TOTAL はバッチまたはオンラインのモードのどちらでも，処理が可能である。オンライン処理では TOTAL は CICS，TSO，CMS，TCAM，BTAM，IMS／DC，ENVIRON／1，INTERCOMM，TASK／MASTER などを含む主要なすべてのライン・コントロール・システムおよびデータ・コミュニケーション・システムとインタフェイスを持っている。

言語からの独立　TOTAL データ・ベースをアクセスするために使われるアプリケーション・プログラムは，COBOL，PL／1，FORTRAN，ASSEMBLER，ASI－ST，SOCRATES など今日使われているほとんどのプログラム言語で書くことができる。

このようにTOTALでは，年中行事である変換はもはや不要となる。

最適なパフォーマンスとエフィシェンシー

伝統的に，この2つの非常に重要なファクターは，コンピュータをベースとしたシステムでは相矛盾しがちなものである。通常，どちらかを得るためには，もう一方を犠牲にしなければならない。それは次の各点の通りである。

＊ アクセスを速くし，大量の出力をするという条件で，高いパフォーマンスをあげるためには，通常大型機を必要とし，それはオペレーティング・システムを始めとする各種のシステム・ソフトウェアのために大きなコア容量を必要とし，大量のディスク・スペース等を必要とする。

＊ 一方コア容量，ディスク・スペース等を効率よく使うためには通常，パフォーマンスを犠牲にする必要がある。

TOTALによる総合的なデータ・ベースの手法では，コア容量，ディスク・スペース等の使用効率面でマイナスを持たらすことなく，パフォーマンスを最適化できるものと考える。

TOTALは上記に述べてきた各設計上の基準を持っているのであるが，次章に述べるようなこれ以外の機能および利点もたくさん持っている。

第2章　TOTAL 概 説

この章は TOTAL データ・ベースの構造について簡単に述べ，TOTAL の3つの段階について紹介する。3つの段階とは次の通りである。

1　データ・ベースの定義
2　データ・ベースのフォーマッティング
3　データ・ベースのアクセス

情報のレベル

TOTAL データ・ベースには5つの情報のレベルがある。これらのうち4つは図2－1に示されている。

アイテムまたはフィールド

これは識別可能なデータの最小単位である。また，アプリケーション・プログラムにおいて伝統的にデータが定義されてきたレベルでもある。
図の例では，市（CITY）である。

エレメント

1つまたはそれ以上の相関連するアイテムの集合である。図では例として住所があるが，それは県，市，町（STATE，CITY，STREET）アイテムより成っている。エレメントは非常に重要である。これがアプリケーション・プログラムが TOTAL とやりとりするレベルである。すなわち，アプリケーション・プログラムは CALL ステートメントの中で読まれ，書かれ，追加され，削除されるエレメントを指定する。

レコード

エレメントの集合。TOTAL のレコードはレコードの伝統的概念と一致する。図のレコードは，名前，住所，電話（NAME，ADDRESS，TEL）のエレメントより成っている。

データ・セットまたはファイル

レコードの集合。TOTAL データ・セットの特徴については，第3章に説明する。

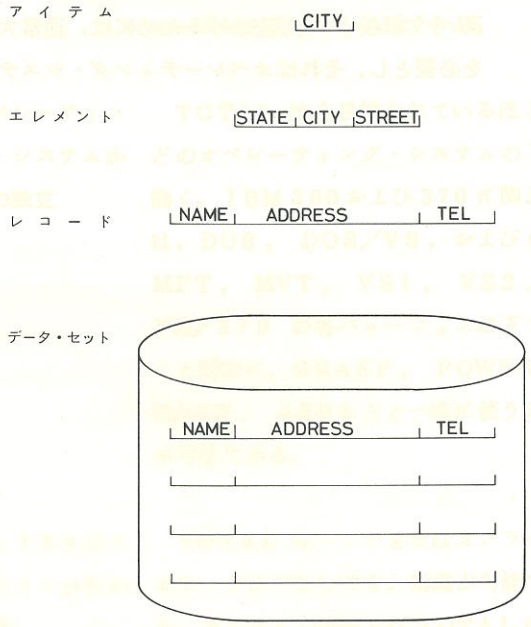


図2－1　情報のレベル(1)

データ・ベース

データ・セットの集合。TOTAL はデータ・ベースの組立てに，ネットワークの手法を用いている。

1つの TOTAL データ・ベースは，最大65,000のデータ・セットを持つことが可能であり，各データ・セットはそのデータ・ベースの中で，2,500 データ・セットまでと直接関連づけることが可能である。TOTAL データ・ベースはほとんど無限の階層レベル，またはネットワーク構造をつくり上げることができる。しかもTOTALはイン

デックス，ディレクトリー，オーバーフロー・エリアなどを必要としないので，オーバーヘッドを気にする必要はない。

データ・ベースの数には制限はなく，どのデータ・セットもいくつものデータ・ベースに含ませることが可能である。

図２－２では，得意先マスター・データ・セットは売掛金データ・ベースおよび受注データ・ベースの両方に所属している。部品マスター・データ・セットは受注データ・ベースと資材明細データ・ベースに所属している。

TOTAL はディスクおよびコアの利用と検索速度の点で，可能な限りの高いパフォーマンスを上げる特徴を持っている。その上データ量が増大し，レコードがデータ・ベース上に追加，削除される場合，よくあるパフォーマンスの低下やファイルの再編成を防ぐために，自己最適化の機能を持っている。

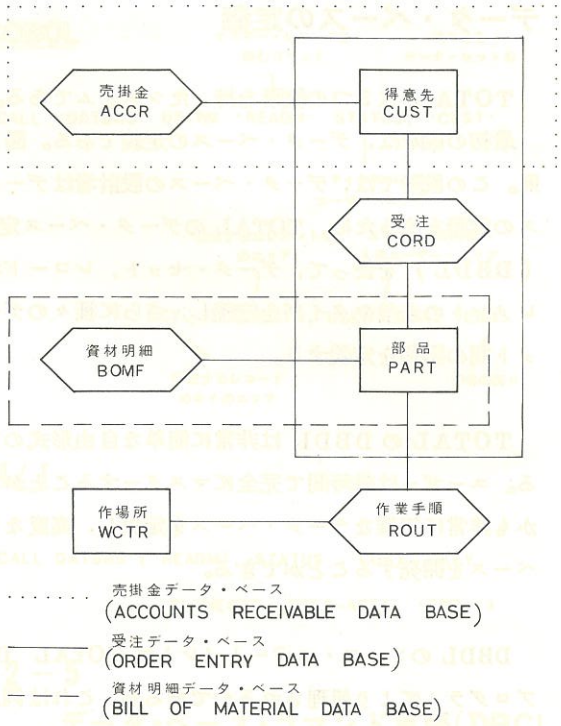


図２－２ 情報のレベル(2)

データ・ベースの定義

TOTAL は3つの段階を持ったシステムである。

最初の段階は，データ・ベースの定義である。図2－3参照。この段階では，データ・ベースの設計者はデータ・ベースの定義をするため，TOTAL のデータ・ベース定義言語（DBDL）を使って，データ・セット，レコードおよびエレメントの名前やタイプを定義し，さらに種々のデータ・セット間の関係を定義する。

TOTAL の DBDL は非常に簡単な自由形式の言語である。ユーザーは数時間で完全にマスターすることができ，しかも非常に複雑なデータ・ベースを定義し，高度なデータ・ベースを開発することができる。

DBDL のソース・ステートメントは TOTAL DBGEN プログラムにより処理されるのであるが，これは例えば，COBOL のコンパイルと同じようなものである。
この DBGEN の出力は次のようなものである。

＊ 診断上の非常に重大なエラーを表示するエラー・マップ（プリント）。

＊ データ・ベースの内容をすべてリストしたデータ・ベース・マップ（プリント）。

＊ アセンブラー・ソースの形でのデータ・ベース定義モジュール：DBMOD（カード，テープ，またはディスク）。

このDBMODは次にアセンブルされユーザー・プログラム・ライブラリーにカタログされる。そして，アプリケーション・プログラムがデータ・ベースをアクセスするときには，いつでも呼び出される。従って，アプリケーション・プログラム上でいちいちファイルの定義を行うことは不要となる。

データ・ベースの全部または一部を変更することが必要となったときには，DBMODの変更部分を定義し，その部分のコンパイル，アセンブル，カタログを行うことが必要となるだけである。アプリケーション・プログラムは自分のプログラム内にファイル定義を持つのではなくて，このモジュールを呼び出すのであるからプログラムの再作成，再コンパイル を必要としない。これがTOTAL がデータからの独立性を保つ1つの方法である。

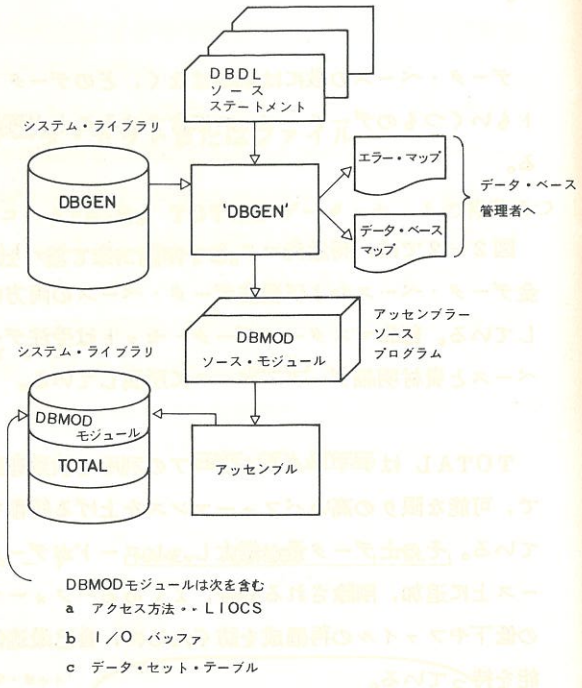


図2－3 データ・ベースの定義手順

データ・ベースのフォーマッティング

TOTAL の第二段階は データ・ベースのフォーマッティングである。図2－4参照。この段階においては，各TOTAL データ・セットのためにとられた DASD エリアが，データ・ベース定義で指定された通りの条件に従って，物理的に，フォーマッティングされる。

TOTAL FORMAT プログラムは，TOTAL データ・セットを格納するエリアをブランク・レコードで埋め，必要なコントロール・レコードをすべて用意することによって当該エリアをフォーマッティングする。

ユーザは使いたい DBMODと，フォーマッティングしたいデータ・セットの名前を，フォーマット・コントロール・カード上に指定する。このDBMODが FORMAT プログラムにリンクされ，フォーマッティングされる各データ・セットについて必要な物理的情報を用意する。

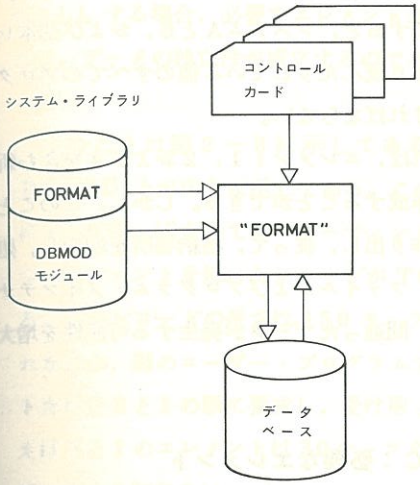


図2－4

データ・ベースのフォーマッティング

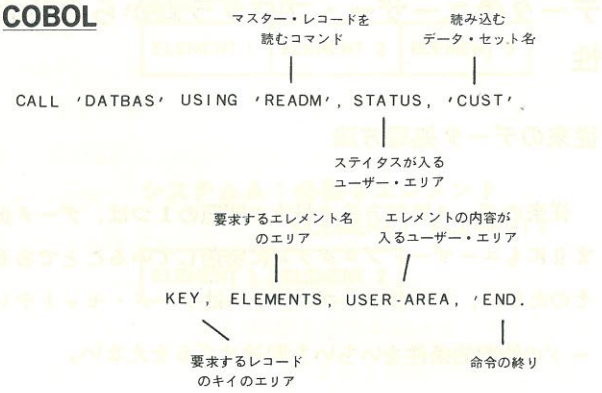
データ・ベースのアクセス

TOTAL の第三段階はデータ・ベースのアクセスである。データ・ベースが既に定義されており，フォーマッティングされている場合，アプリケーション・プログラムにデータ・ベースを結びつけるには TOTAL のデータ・ベース・コマンド言語（DBCL）を使う。これは，例えば，レコードを読んだり，書いたり，追加したり，または削除したりする場合である。

TOTAL DBCLコマンドは，ユーザー・プログラムの中のCALLステートメントを経由して出され，サブルーチンCALLステートメントをサポートするプログラミング言語，例えばCOBOL，FORTRAN，PL／1，ASSEMBLER，ASI－ST，SOCRATESなどと一緒に使用することができる。

COBOL や PL／1に使用される DBCL コマンドの例を図2－5に示す。

COBOL



PL/I

```
CALL DATBAS ('READM', STATUS, 'CUST', KEY,
     ELEMENTS, USER-AREA, 'END.')
```

図2－5

データ・ベース・コマンド言語（DBCL）
ステートメント例

この例では，CUST マスター・データ・セットからレコードを読んでいる。CALL を実行する前に KEYの位置に読み込みたい特定レコードのキイを入れ，さらにパラメータ ELEMENTS の位置に読み込みたいエレメント名を入れるのである。TOTAL はオペレーションがうまくいった場合には，STATUS に＊＊＊＊のメッセイジを出し，USER－AREAに要求したエレメントを書き出す。オペレーションにエラーのあった場合には，TOTAL はSTATUSに診断メッセィジを書き出す。

CALL が完了すると，ユーザー・プログラムにコントロールが戻る。

データのユーザー・プログラムからの独立性

従来のデータ処理方法

従来のデータ処理方法の最大の問題の1つは，データがあまりにもユーザー・プログラムに密着していることである。そのために，ユーザー・プログラムはデータ・セットやレコードの物理的条件をいちいち記述せざるをえない。

典型的な例が図2－6に示されている。3つのエレメントをもつレコードがある。システムAはエレメント1と2しか使わないとしても，全レコードを読まなければならない。システムBはエレメント2と3しか使わないとしても，全レコードを読まなければならない。

基本レコード

ELEMENT 1	ELEMENT 2	ELEMENT 3

システムA：必要なエレメント

ELEMENT 1 , ELEMENT 2

ELEMENT 1	ELEMENT 2	ELEMENT 3

システムB：必要なエレメント

ELEMENT 2 , ELEMENT 3

ELEMENT 1	ELEMENT 2	ELEMENT 3

図2－6 従来の方法：システムAとB

さて新しいシステム，システムC，を作成する必要が生じ，エレメント1と2の他に，現在基本レコードに含まれていない，新しいエレメント4が必要となったとき，どうなるであろうか。2つの方法がある。図2－7参照。

1. エレメント4を基本レコードに加えることができる。しかしそのようにすると，システムAとB，および基本レコードを更新したり読んだりしている他のすべてのプログラムを変更しなければならない。
2. そうでなければ，エレメント1，2および4を含む新しいレコードを作成することができる。しかし，このことは重複データを作り出し，従って，格納場所を増やし，処理の効率を落し，ファイルおよびプログラム・メインテナンスを複雑にし，間違ったデータが発生する可能性を増大させる。

システムC：必要なエレメント

ELEMENT 1 , ELEMENT 2 , ELEMENT 4

第1の方法：基本レコードを拡張する

ELEMENT 1	ELEMENT 2	ELEMENT 3	ELEMENT 4

結果：システムAとBを変更しなければならない

第2の方法：新しいファイルを作る

ELEMENT 1	ELEMENT 2	ELEMENT 4

結果：重複データを生じ
処理時間が増え
メインテナンスが複雑になり
データが不正確になる

図2－7 従来の方法：システムC

TOTALのデータ処理方法

TOTALは，データとそのデータをアクセスするアプリケーション・プログラムとの間に，高度の独立性を持たせることにより，このような問題を解決している。

成長や変化に対応できるモジュール構造と発展性を持った手法を実現させるためには，データの独立性が重要なこととなる。アプリケーション・プログラムが物理的にデータに依存する関係は，どのようにわずかなことでも取り除かなければならない。

TOTALは，ユーザー・プログラムがTOTALにCALLする場合，必要なエレメント名だけを指定しているので，データの独立性が成立するのである。

このことは図２－８に示してある。TOTALとDBMODとが中央に示してある。この例では(1)のCALLによって(3)，(4)を経てオペレーティング・システムにより５つのエレメントを持ったレコードがTOTALに読み込まれる。このレコードの長さは150キャラクターであるかもしれないが，図のユーザー・プログラムでは１と３のエレメントだけを３と１の順に要求し，受け取っている。従って，例えば３と１のエレメントは30キャラクターだけで構成されているかも知れない。

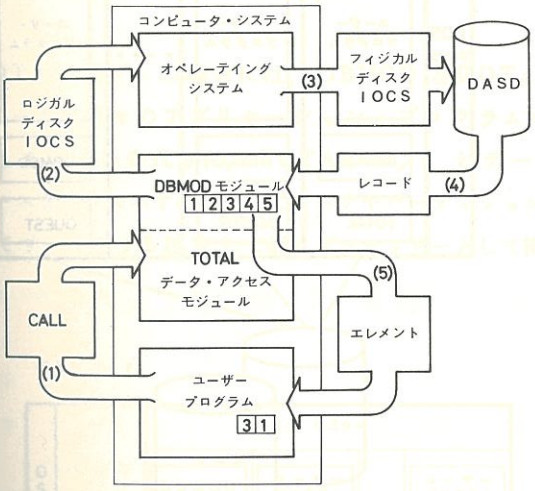


図２－８

エレメントを要求してから受け取るまで

このような方法によって，ユーザー・プログラムは求めるデータのエレメントだけに関連を持ち，レコードやファイルの物理的な特性とは何らの関連も持つ必要がなくなっている。

従って最初に掲げた例におけるシステムAとBは，それぞれのアプリケーションに必要なエレメントのみを読むCALLを行う。TOTALはCALLに指定されたエレメントのみを抽出し，それらをアプリケーション・プログラムに渡す。図２－９参照。

基本レコード

ELEMENT 1	ELEMENT 2	ELEMENT 3

システムA：必要なエレメント

ELEMENT 1 , ELEMENT 2

ELEMENT 1	ELEMENT 2

システムB：必要なエレメント

ELEMENT 2 , ELEMENT 3

ELEMENT 1	ELEMENT 3

TOTALは要求されたエレメントのみを抽出し、ユーザー・プログラムへ渡す

図２－９

TOTALの方法：システムAとB

新しいシステムの開発が基本レコードの拡張を必要とする場合には，既存のシステムに何ら影響を与えることなく行うことができる。すなわち，再プログラムも再コンパイルも必要としない。

エレメント4を基本レコードにつけ加えるためには，そのデータ・セットのDBMOD を書き直し，再コンパイルし，再カタログし，かつデータ・セットを再ロードしなければならない。しかしエレメント1，2，3のみをアクセスする20本，またはおそらく200本のアプリケーション・プログラムは何の変更も必要としない。図2－10参照。

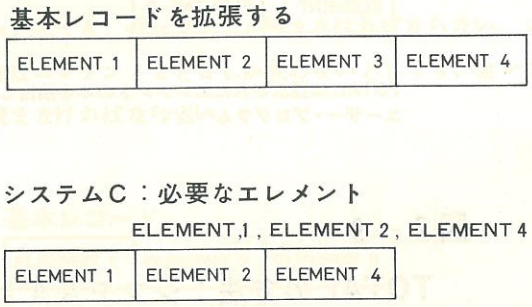


図2－10 TOTALの方法：システムC

TOTAL 4

TOTAL 4は IBM DOS および OSオペレーティング・システムの下で，単一タスクまたは複数タスクのバッチ処理に用いられる。この中にはこれらのオペレーティング・システムのVS バージョンも含んでいる。図2－11参照。

TOTAL 4では，DBMODとTOTAL の両方がユーザーのアプリケーション・プログラムとして，同一のパーティションにある。

DBMODは，データ・セット当り約200バイトのコアとI／Oバッファ・エリア（ユーザーが指定する）を必要とする。

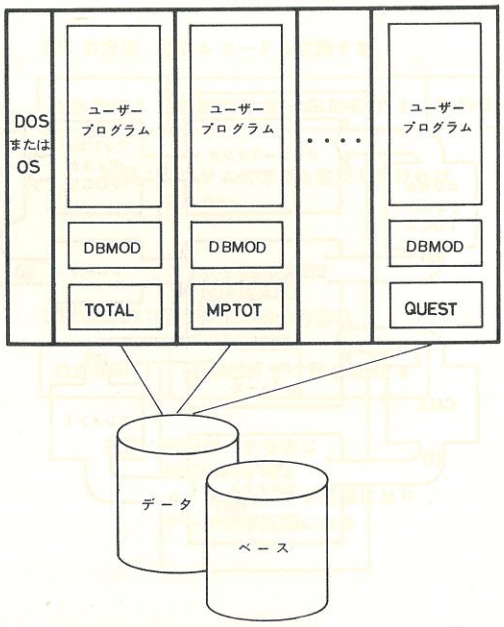


図2－11 TOTAL 4

オペレーション・モードとデータ・ベースの保護

TOTAL にはいくつかの別なバージョンがあり，各々が異なったオペレーション上の環境で使われるようになっている。

TOTAL のモジュールはすべてのデータ・マネジメントの機能を実行する。例えば読み込み，書き込み，追加，削除といった各機能である。8Kバイトのコアを必要とする。TOTAL を使う代りに，次の TOTAL サブセットの1つを使うことにより，使用するコアを節約することができる。

MPTOT …… 読み込みおよび書き込みの機能のみを実行できる。5Kバイトのコアを必要とする。

QUEST …… 読み込み機能のみを実行できる。3Kバイトのコアを必要とする。

複数のパーティションでアプリケーション・プログラムが同じデータ・ベースを処理しているときには，TOTAL 4はレコードが同時に更新されるのを防ぐ機能を持っている。

TOTAL 4には，データ・ベース処理に関するロギングの機能は何も備えていない。

TOTAL 5／6

TOTAL 5／6 は IBM OS および OS／VS オペレーティング・システムの下で，バッチ処理マルチ・プログラムとオンライン処理マルチ・タスクとに用いられる。図2－12参照。

TOTAL 5／6 の場合には，DBMOD と TOTAL モジュールは個々のアプリケーション・プログラムにリンクしない。その代りに，TOTAL 5／6 はデータ・ベースをアクセスするすべてのタスクやパーティションに対してサービスする独立して働くプロセッサーとして機能する。

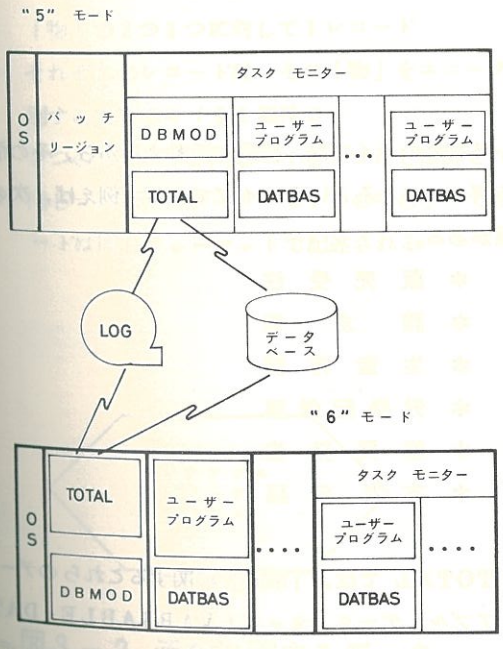


図2－12 TOTAL 5/6

TOTAL 5／6 は，2つのモードによるオペレーションのどちらかに使用される。

"5"モード …… TOTAL はサブ・タスクとしてアタッチされ，タスク・モニターの下で，テレプロセシング・リージョンのすべての他のサブ・タスクに対してサービスする。

"6"モード …… TOTAL は1つのリージョンまたはパーティションの中の1つのプログラムとして働き，当該コンピュータ・システムのすべてのタスクまたはサブ・タスクにサービスする。

上記のどちらの場合でも，TOTAL のタスク・リージョンまたはパーティションは次のコア・スペースを必要とする。

TOTAL	14Kバイト
DBMOD	1データ・セット当り200 バイトと I／O バッファ・エリア

DATBAS と呼ばれるインタフェイス・モジュールは，1,400 バイト必要で，これは TOTAL データ・ベースをアクセスする各ユーザーのアプリケーション・プログラムにリンク・エディットされる。

データ・ベース操作についてユーザーに許されている諸機能をすこしも制限することなく，TOTAL 5／6 は，データ・ベースを集中的に管理している。そして多くのタスクおよびサブ・タスクが共通のデータ・ベースを同時にアクセスし，更新することができる。

データ・ベースの保全機能は，データ・ベースに加えられる種々の変化に関する，TOTAL 5／6 の集中化された時系列的ロギング機能によって行われる。

第3章　データ・ベース構造

この章は，TOTAL の最も重要な部分 ── そのデータ構造を紹介する。

データ構造の特徴

TOTAL の独特なネットワーク・データ構造は，

* データ・ベース中のデータのどの項目に対しても，多くのエントリー・ポイントを与えることができる。すなわち，データのどの項目でも，ユーザーが定義したいくつものキイでアクセスすることができる。
* データ・ベースの中のデータのどの項目でも，これを他のデータのどの項目にも，関連づけることができる。
* 重複するデータを排除できることによって，記憶装置と処理時間を節約することができる。
* データ・ベースをシリアルでも，ランダムでも，非常に効率よく処理することができる。
* 情報処理要求が変った場合でも，現存するアプリケーション・プログラムにほんの少ししか，または全く影響を与えることなく，データ・ベースを容易に変更することができる。

要約すると，このデータ構造により TOTAL ユーザーは，磁気テープ指向型の大量順次処理システム ── これは本質的には，昨日の活動に関する歴史的レポートを作成するシステム ── を，きょうの活動を監視し，あすの活動を計画するのに必要な，最新の情報を常に備えている総合データ・ベース・システムにかえることができる。

どのような会社のデータでも，これをはっきり区別できる2つのタイプに分類することができる。

最初のタイプは，物が存在するから，その物に関するデータも存在するというタイプである。例えば，次の情報がある。

* 従　業　員
* 得　意　先
* 生産または販売する商品
* 購買する部品または資材
* その設備，たとえば，
  - 子　会　社
  - 工　　場
  - 本　　社
  - 支　　店
  - 車　　両

TOTAL では，「物」に関するこれらのデータを，マスター・データ・セット（MASTER DATA SETS）と呼ぶものに構成する。マスター・データ・セットは図示するときは四角であらわす。図3－1参照。

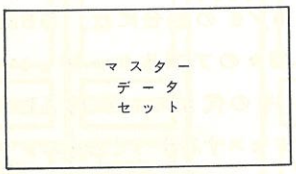


図3－1　物に関するデータ

2番目のタイプは，組織が活動するから，その活動に関するデータがあるというタイプである。例えば，次のような情報がある。

* 販　売　受　注
* 請　　求　　書
* 生　産　指　図
* 労働記録票
* 購　買　発　注
* 支　払　伝　票

TOTAL では，「活動」に関するこれらのデータを，バリアブル・データ・セット（VARIABLE DATA SETS）と呼ぶものに構成する。バリアブル・データ・セットは六角形であらわす。図3－2参照。

「物」に関するデータ

「物」に関する情報について一般的に次のようなことがいえよう。

1　すべての「物」はそれぞれユニークである。すなわち，個々の得意先，従業員または商品はそれぞれユニークな存在である。ユニークであるがために，それを記述するユニークな記録（レコード）が存在しなければならない。

2　ユニークな「物」を，ユニークに識別するために，それぞれの得意先，従業員，商品等を識別するユニークな名前または番号をもつ。

3　ある1つのグループの中の1つの「物」――例えば得意先グループの中のある顧客――を記述するのに必要な情報は，同じグループの中ではすべて，他の「物」――別の顧客――を記述するのに必要な情報と同じである。つまり1つのグループは同じ情報項目で表現される。

TOTAL はそれぞれのタイプの「物」を記述するために，それぞれにマスター・データ・セットを作る。例えば，得意先のための得意先マスター・データ・セット，従業員のための従業員マスター・データ・セット，商品のための商品マスター・データ・セット等である。

それぞれのマスター・データ・セットは，次のような特徴をもっている。図3－3参照。

＊　「物」の1つ1つに対して1レコード
＊　それぞれのレコードは，その「物」をユニークに識別するためのキイを1個持つ
＊　2つのレコードが同じキイをもつことは許されない
＊　同じマスター・データ・セットの中のすべてのレコードは同じフォーマットで記述される

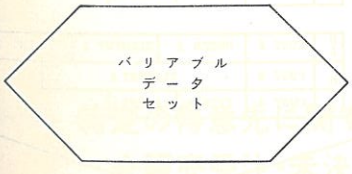


図3－2　活動に関するデータ

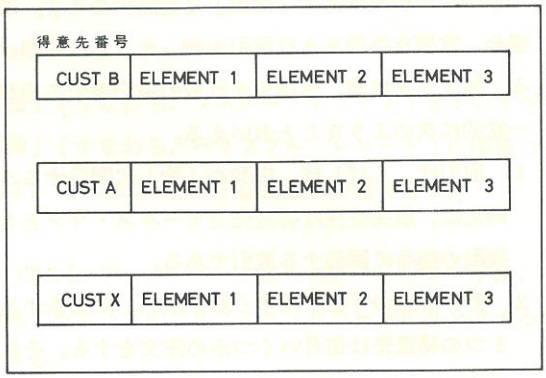


図3－3　マスター・データ・セット

「活動」に関するデータ

すべての企業はその「活動」を記録にのこす。ほとんどの場合，重要な活動または取引が起ったときに書類が作成される。図3－4参照。活動または取引を記録する書類について，一般的に次のようなことがいえる。

1. 取引は，しばしば，複数の「物」に関係することがある。例えば，販売受注は得意先とセールス・マンと1つまたは複数の商品に関係する取引である。
2. それぞれの「物」は多くの「取引」に関係する。例えば，1つの得意先は毎月いくつかの注文をする。それぞれの注文はいくつかの商品に関係している。
3. 1つの「取引」に関する書類は，いくつかの種類の情報を含んでおり，それぞれその種類によって異ったフォーマットを必要とする。さらに，それを記述する情報の量――例えば字数，行数――は，それぞれの「取引」によって異なる。

従って，TOTAL バリアブル・データ・セットは，次のような特徴をもっている。図3－5参照。

＊ それぞれのレコードに複数のキイがある。それぞれのキイは，取引に関係する「物」の1つ1つを識別する。

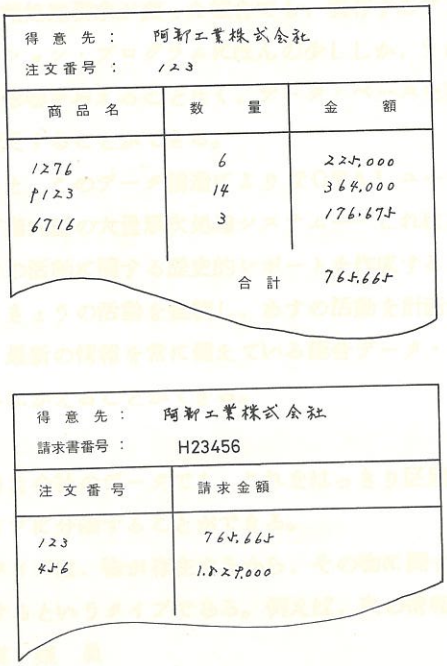


図3－4 企業活動の記録

＊ それぞれのキイに複数のレコードがありうる。それぞれのレコードは，ある「物」に関する1つの取引をあらわす。
＊ 複数のフォーマットが許される。バリアブル・データ・セットは，いくつかの異ったタイプのレコードをもつ。そして，それぞれが異ったフォーマットをもっている。

「物」と「活動」の関係づけ

今までのところ，TOTAL がいかにデータを構成するかを説明してきた。我々が本当に興味があるのは，いかに我々がこれらのデータを使うことができるかである。

前に「要求にこたえる情報提供」について説明した。……すなわち情報要求を持つ人がいかに迅速に容易に必要情報を選択し抽出することができるかということである。

近代的企業にとって，その活動を計画し管理するため中枢神経の役割をはたすマネジメント・インフォメーション・システムにおいては，ユーザーは次のような情報を検索する場合が多いと思う。

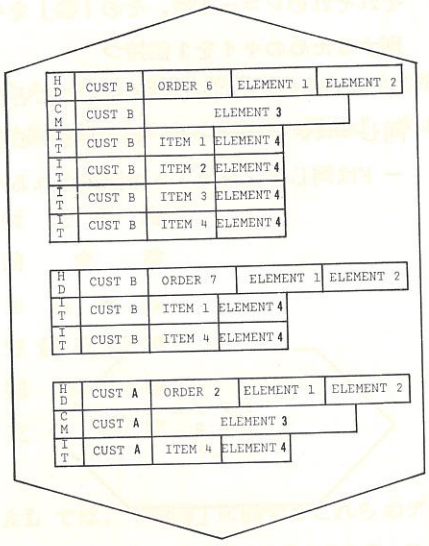


図3－5 バリアブル・データ・セット

＊ 特定の得意先に関する全販売受注
＊ 特定の商品に関する全販売受注
＊ 特定の日に受けた全販売受注
＊ 特定の日に出荷される全販売受注
＊ 特定の得意先に関する未決済請求書
＊ 特定の日に送られる全請求書
＊ 送られてから，例えば，30日以内に支払われていない全請求書

いうまでもなく，これらの要求はすべて，ある「物」に関係した，ある「活動」に関する情報である。図3－6参照。

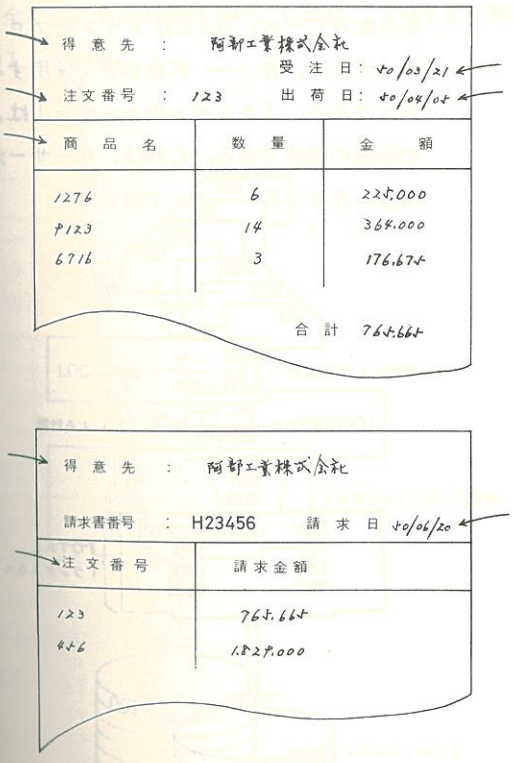


図3－6 特定の得意先に関する全販売受注・未決済請求書

TOTAL では，これらの要求は，特定のマスター・レコードに関連する一群のバリアブル・レコードを検索することによって満たされる。図3－7参照。

「物」（すなわち，マスター・レコード）をそのすべての「活動」（すなわち，バリアブル・レコード）に関係づける機能は，TOTAL データ構成法の中核である。

TOTAL データ・ベースのマスター・レコードは，それが関連するすべてのバリアブル・レコードにリンクすることができる。

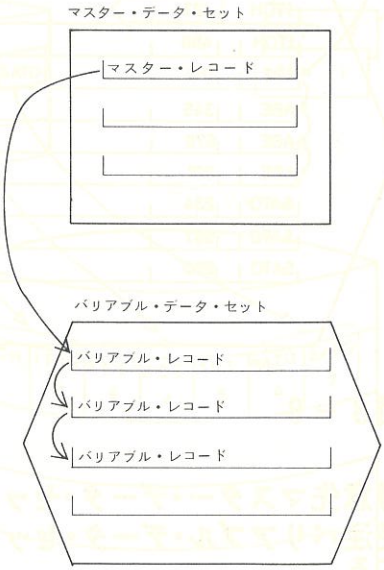


図3－7
マスター・データ・セットからバリアブル・データセットへリンクする

例えば，得意先マスター・レコードの中のある顧客との取引関係は，販売受注バリアブル・レコードの中の当該顧客の全取引記録とリンクすることにより表わすことができる。図３－８参照。

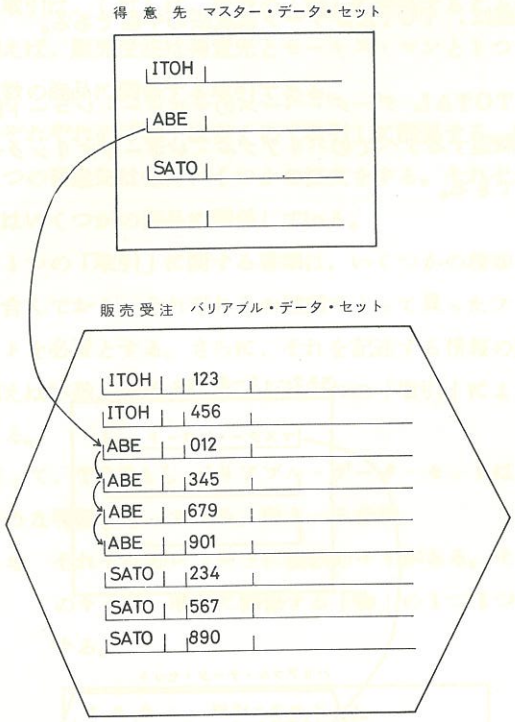


図３－８

得意先マスター・データ・セットから販売受注バリアブル・データ・セットへリンクする

マスター・レコードの格納方法

ランダム化

マスター・レコードをユニークに識別するキイは，当該レコードを直接アクセスするのに使用される。TOTALはこのキイからマスター・レコードの記憶番地（アドレス）を計算し，そのレコードをアクセスするのに直接そのアドレスに行くことができる。図３－９参照。

キイからマスター・レコードのアドレスを計算するTOTALの「ランダム化」ルーチンは，非常に効率がよい。テストの結果，たとえマスター・データ・セットが，割りあてられた記憶容量の90％までロードされているとしても，TOTALは，95％は１回のシーク（SEEK）と１回のリード（READ）で，当該マスター・レコードをアクセスできることがあきらかになっている。この結果を２回のシークと２回のリードを最小限必要とするISAMまたはVSAMと比較していただきたい。

TOTALはランダム化機能により，たとえ，どんなに頻繁に情報が更新追加，削除，されても，データ・セットを再編成することなく，高い実行効率を維持することができる。

注：「ランダム化ルーチン」は，データ・セットをその物理的特徴 …… 例えば，キイの長さ，割りあてられたスペース，DASDタイプ，ブロッキング・ファクター，等 …… に合わせて使用する汎用ルーチンである。「ランダム化ルーチン」は，TOTALの必須の部分である。これは，ユーザーが書いたり，用意したりするルーチンではない。

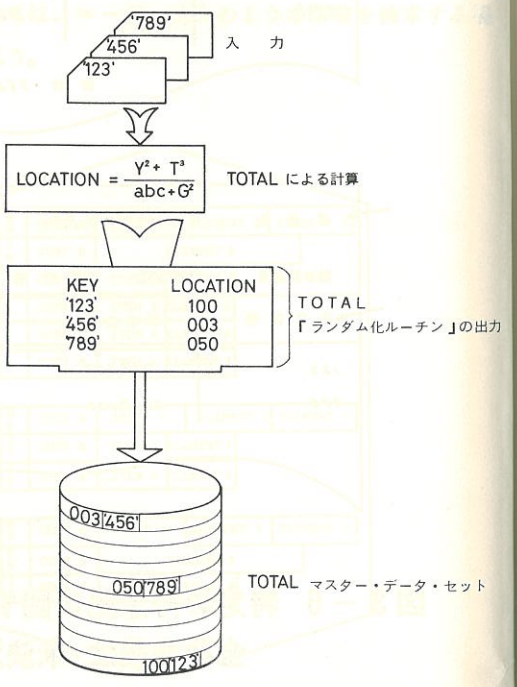


図３－９ ランダム化ルーチン

シノムニ

ロード率90％の場合に，1回のシークと1回のリードで，95％はアクセスできると述べたことを思い出してほしい。それでは，残りの5％はどうなのか？

1回でアクセスできない場合は，「シノニム（synonym）」がある。図3－10参照。例えば，TOTALのランダム化ルーチンがレコード345とレコード123に対して番地計算の結果同じ相対番地（relative location）になったとしよう。

TOTALは，シノニム・レコードをできるだけこの計算結果による番地（ホーム・レコード）に近く …… 同じブロック，同じトラック，同じシリンダー …… におき，ホーム・レコードと「シノニム」のチェインをつけておく。

もちろん，アプリケーション・プログラムは，シノニムの存在を全く気にする必要はない。これは，TOTAL独自の仕事の1つである。

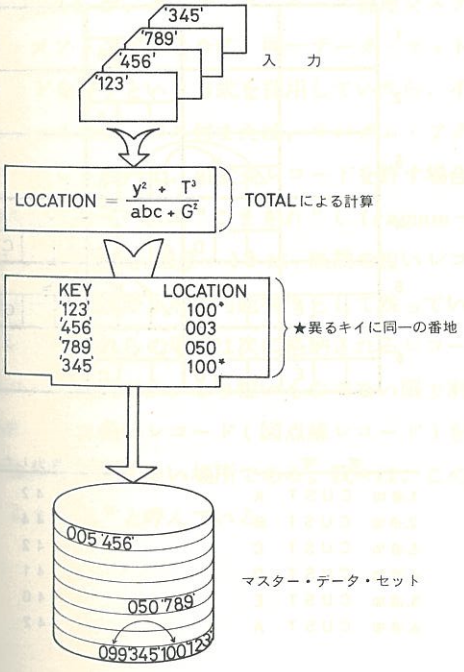


図3－10 シノニム(SYNONIM)

バリアブル・レコードの格納方法

バリアブル・レコードはDBMODで，定義されたレコード長やブロック・サイズに従い，データ・セットの最初のブロックからディスク・アドレスの順に格納される。

図3－11参照。

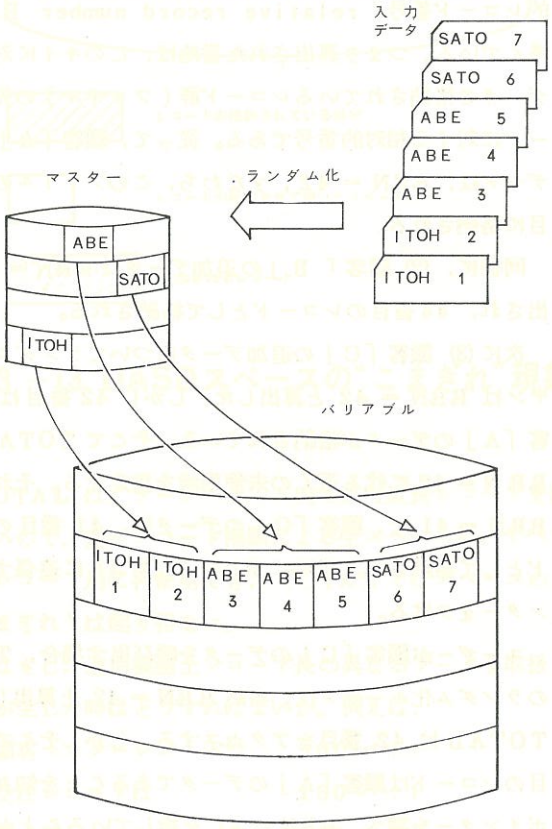


図3－11 バリアブル・レコードの格納

DASDスペースの管理

図3－12はあるマスター・データ・セットにレコードが追加されたり，削除が行なわれたりした場合にTOTALがDASDスペースをいかに効率よく利用するかを例示するものである。

まず (1) 顧客「A」のレコードをあるデータ・セットに追加する場合。

TOTALのランダム化ルーチンは，このデータ・セットのキイからある計算をして，このレコードを格納すべき相対番地を算出する。この相対番地は，物理的番地というよりはむしろ論理的番地であるというべきで，我々は，これを相対的レコード番号（relative record number RRN）と呼んでいる。つまり算出された番地は，このキイに対応するデータの格納されているレコード群（ファイル）の先頭レコードに対する相対的番号である。従って，顧客「A」の追加データは，RRN＝42，すなわち，このファイルの42番目に格納される。

同様に，(2) 顧客「B」の追加データはRRN＝44と算出され，44番目のレコードとして格納される。

次に(3) 顧客「C」の追加データについてランダム化ルーチンはRRN＝42と算出した。しかし42番目は既に顧客「A」のデータが格納されている。そこでTOTALは，RRN＝42に代る近くの未使用域を指定する。それがRRN＝41で，顧客「C」のデータは，41番目のレコードとして格納され，同時にRRN 42を41に連係するポインターをつける。

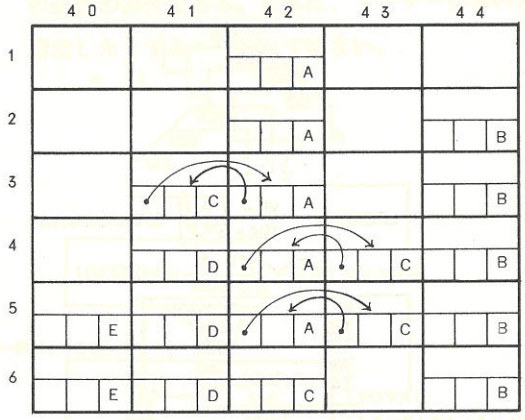
ユーザーが顧客「C」のデータを呼び出す場合，TOTALのランダム化ルーチンは，当然RRN＝42と算出し，TOTALは42番目をアクセスする。しかしそこで42番目のレコードは顧客「A」のデータであることを知り，そのポインターを調べ，RRN＝41を指していることから，41番目のレコードをアクセスして，「C」のデータを得ることができる。ほとんどの場合RRN 42と41は，共に同じブロック内に …… 従って又同じバッファ内に格納されているから，1度RRN＝42をアクセスすれば，顧客「C」のデータをアクセスするために改めてI/Oオペレーションを行なう必要はない。

(4)顧客「D」のレコードが追加される場合。TOTALのランダム化ルーチンは，RRN＝41を算出した。そこには顧客「C」のデータが既に格納されている。この場合に，顧客「C」のデータはもともとRRNは41ではなく，42に属し「シノニム」として41に格納されていたものである。そこでTOTALは顧客「C」のデータをRRN＝42に一番近い未使用域（RRN＝43）へ移し，顧客「D」のレコードをキイから算出したRRNどおり41番目に格納する。

(5) 顧客「E」のレコード追加は，ランダム化ルーチンはRRN＝40と算出し，TOTALは，そこに「E」のデータを格納する。

(6) TOTALが管理するファイルからあるレコードが削除されると，当該スペースは，即時に解放され利用可能域になる。顧客「A」のレコードが削除されると，TOTALはRRN＝42を即刻解放し，再使用できるようにするのであるが，顧客「C」のレコードがもともとRRN＝42でありながら，シノニムで，RRN 43にある。この場合TOTALは「C」を42に移し，43を解放する。

バリアブル・データ・セットからレコードを削除する場合，TOTALはそのスペースを即時解放し，再使用できるようにする。レコードを追加する場合は，関連するレコードの近く（同じブロック，同じトラック，同じシリンダーの順に空きスペースを捜す）に格納する。



処　理	キイから算出した　RRN
1.追加　CUST　A	42
2.追加　CUST　B	44
3.追加　CUST　C	42
4.追加　CUST　D	41
5.追加　CUST　E	40
6.追加　CUST　A	42

図3－12
相対的レコード番号
(relative record number ‥ RRN)

DASDスペースを上手に管理することは，次の理由からも重要なことである。

＊ データ格納に要するDASDスペース総量を節約する。

＊ データを所定の場所またはその近く ……例えばシノニムのように計算された場所の近く …… に格納する。従って，同一バリアブル・チェインに属するすべてのレコードがある場所に集められていることは，アクセス・タイムを最小化する。

図3－12は，TOTALのDASDスペース管理の重要な一面を示したものであるが，TOTALにはもう1つの重要なスペース管理上の特徴がある。それは，TOTALでは，同じデータ・セットに格納されるレコードはそのレコード長が同じ長さに統一されるということである。図3－12に示すとおり，どのレコードが削除されても，その場所は即時に解放され別のレコードのために使用可能となる。即ち，新しく追加されるレコードであれ，シノニムで別の場所に格納されていたレコードであれ，どんなレコードでも解放された場所に，ちゃんと納めることができる。それは，同じデータ・セットに属するレコードはすべて同じ長さだからである。これはもしTOTALが，あるデータ・ベース管理システムや，アクセス・メソッドのように，同一データ・セット内で可変長レコードを許すという方式を採用していたら，不可能なことである。インデックス付または，ランダム・アクセス式のデータ・セットの内で，可変長レコードを許す場合はいつでも，DASDスペースに"こまぎれ"（fragmmentation）現象が起る。図3－13は，無数の短いレコードが削除され（図斜線部分）利用可能場所として残っていることを示している。これらの場所は次に格納されるレコードが必ず前のレコードと同じか，より短いものでない限り利用できないために，より長いレコード（図点線レコード）を格納しようとする時は，使えない場所である。我々は，このような状態を"こまぎれ"と呼んでいる。

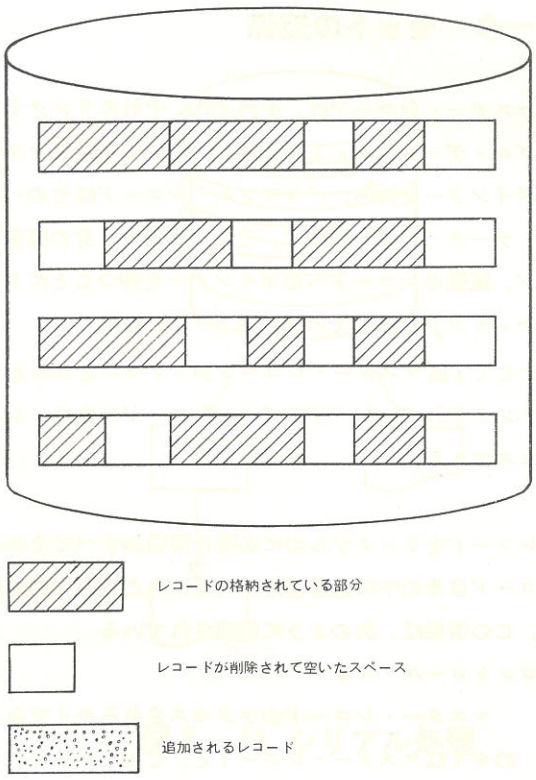


図3－13 DASDスペースの"こまぎれ"現象

TOTALは，データ・セット内では可変長レコードを許さないので，逆にレコード削除による空スペースは，すべて，他のレコード用に再使用でき，TOTALではスペースの"こまぎれ"は起り得ない。

ではもし，適用業務上レコード長の異なるデータを取扱う必要が生じた時はどうすればよいか。例えば，

顧客マスターレコードは　300バイト
受注レコードは　150バイト
請求書レコードは　75バイト

とする。

TOTALは，単純にこれら3つの長さの異るレコードごとに，3種類のデータ・セットを設定するに過ぎない。すなわち，300バイト長のレコードのデータ・セット
150バイト長のレコードのデータ・セット
75バイト長のレコードのデータ・セット
そして，TOTALの，ネットワーク・データ構造機能を使って，3つのデータ・セットを互に連係するのである。

データ・セットの連係

マスター・レコードは，それぞれ，それにリンクするバリアブル・データ・セットの先頭レコードと最後尾レコードへのポインターを持ち，バリアブル・レコードはそのバリアブル・データ・セットの中で，それぞれ自分自身の直前のレコード，直後のレコードへのポインターを持つことによって「チェイン」を作っている。図3－14 参照。

そこで1度マスター・レコードがアクセスされると，それにリンクされているバリアブル・レコードの全チェインがアクセスできる。

レコードをリンクするのに必要な情報がすべてそれぞれのレコード自身の中に直接蓄えられていることが大切な点である。この情報は，次のように構成されている

コントロール・キイ

マスター・レコードがアクセスされるキイである。このキイはマスター・レコードと，そのマスター・レコードにリンクするすべてのバリアブル・レコードがもっている。キイの長さはユーザーが指定する。

リンケージ・パス

これは8バイト長の1フィールドを占め，マスター・レコードとそのマスター・レコードにリンクするそれぞれのバリアブル・レコードがもつ。

マスター・レコードのリンケージ・パスは，リンクするチェインの先頭のバリアブル・レコードに対する4バイトのポインターと，そのチェインの最後尾のバリアブル・レコードに対する4バイトのポインターを含んでいる。バリアブル・レコードのリンケージ・パスは，チェインの中の前のバリアブル・レコードに対する4バイトのポインターと，次のバリアブル・レコードに対する4バイトのポインターを含んでいる。従って，チェインの先頭のバリアブル・レコードのリンケージ・パスの前の4バイトと，最後尾のレコードのリンケージ・パスの後の4バイトはブランク表示となる。

TOTAL はリンク情報のこのような構成により，いわゆるインデックスとか，テーブル，ディレクトリー等の特別な連係手段を必要としない。

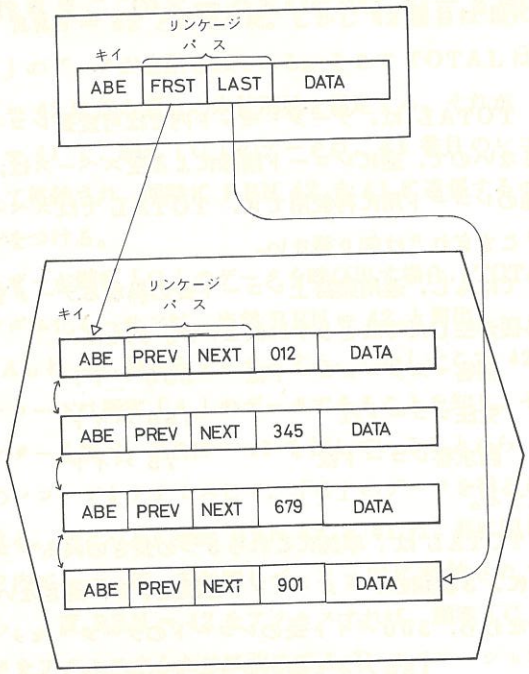


図3－14

マスター・データ・セットからバリアブル・データ・セットへリンクする

順　次　処　理

「物」または「活動」に関する情報をアクセスまたは処理する方法の1つにシークェンシャル処理法がある。例えば，全得意先，全販売受注，全請求書または全商品に関する順次レポートを作成することがある。

もちろん，これは，過去の歴史的レポート指向型のシークェンシャル・テープ・システムを代表する種類の処理方法である。図3－15 参照。

これもまた，TOTAL で実行可能である。なぜならば，

* TOTAL データ・セットは，インデックス，テーブル，ディレクトリー，オーバーフロー・エリア等は何も持っていない。必要情報はすべてユーザーが定義したレコード自身が持っている。
* バリアブル・データ・セットは，それが関連するマスター・レコードのキイ …… すなわち，どのレコードとリンクするかを識別するデータ …… を持っている。

＊ TOTALは，シークェンシャル・アクセス命令を持っている。

ついでながら，このタイプの順次処理は，TOTALでは非常に効率がよい。ISAMまたはVSAMのようなファイルのシークェンシャル処理よりは，はるかに効率がよい。

検索のみならず，TOTALはデータ・セットに対して，レコードの追加，削除，更新等をシークェンシャルにバッチ処理することもできる。

ランダム処理

例外レポートが経営情報システムの1つの目的であることは前に述べた。

コンピュータに次のようなことを行なわせるプログラムは，その1つの例ということができよう。

1. 新しい販売受注があったとき，注文を発した得意先について滞っているすべての未決済請求書をチェックする。
2. もし，その得意先に「支払期日経過」請求書がなければ，注文をうける。
3. もし，得意先に「支払期日経過」請求書がある場合には，注文をことわり，この注文の内容の他に，その得意先のマスター・レコードからの情報，その得意先のすべての「支払期日経過」請求書からの情報を含む報告書をプリントする。

このためには，それぞれの得意先のすべての請求書を含むバリアブル・データ・セットが，得意先マスター・データ・セットにリンクされていなければならない。

図3－16は，1つの得意先マスター・レコードが，すべての受注およびこの得意先に関連する請求書バリアブル・レコードにリンクするチェインを示している。

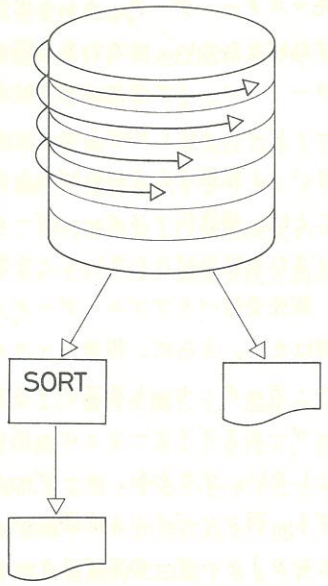


図3－15 シリアル処理

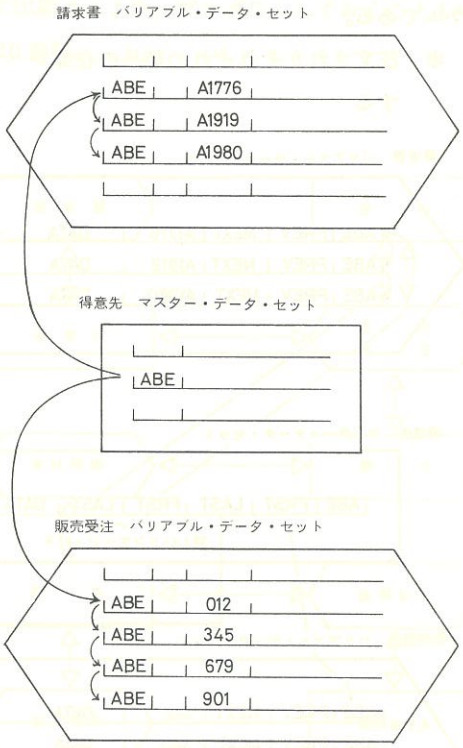


図3－16 請求書―得意先―販売受注

請求書チェインをマスター・レコードにリンクするためには，得意先マスター・データ・セットに変更が必要なことに注意しなければならない。すなわち，新しいリンケージ・パスがマスター・レコードに追加されなければならない。得意先マスター・レコードはこの新しいリンケージパスを収容するために8バイト拡張されなければならない。図3－17 参照。このことは，得意先マスター・データ・セットが物理的に再ロードされなければならないことを意味している。

しかし，販売受注バリアブル・データ・セットには，何ら変更の必要はない。さらに，得意先マスターまたは販売受注バリアブル・データ・セットをアクセスするために以前に書かれたユーザーのアプリケーション・プログラムは，変更を全く必要としない。すなわち，ユーザー・プログラムは再コーディングも，再コンパイルも必要ない。それは（前の章で説明したとおり）ユーザーのアプリケーション・プログラムが関係するのは，当該データの中の所要エレメントであって，当該レコードの長さとは無関係だからである。

例外レポートのもう1つの例は，新しい販売受注がシステムに入るたびに，次のことをコンピュータに行なわせるプログラムである。

＊ 注文されたそれぞれの商品の在庫レベルをチェックする

＊ 十分な在庫がある場合には，注文をうけ，在庫レベルを更新する

＊ もし，十分な在庫がない場合には，注文をことわり，注文に関する情報，注文された商品に関する情報，注文をした得意先に関する情報を含むレポートをプリントする。

管理者は，このレポートを使って状況を判断し，適切な決定を下す。

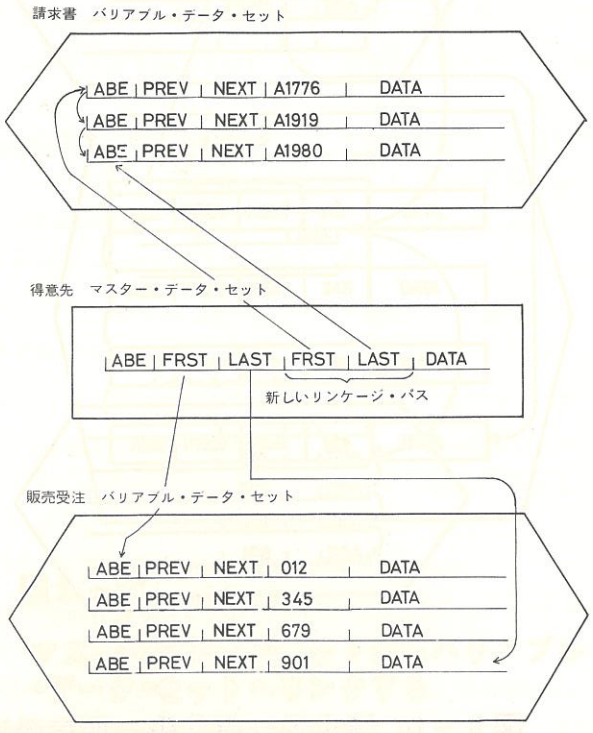


図3－17 請求書―得意先―販売受注

このレベルの例外レポートを実施するには，商品マスター・データ・セットが作成され，販売受注データ・セットにリンクされなければならない。そして，それぞれの受注レコードは，得意先マスター・レコードおよび商品マスター・レコードにリンクされる。

商品データ・セットを販売受注データ・セットにリンクするためには，販売受注レコードは商品データ・セットに対する8バイトのリンケージ・パスを入れるために販売受注レコードを拡張しなければならない。図3－18 参照。しかしこの場合にも

＊ 他のデータ・セット

＊ 以前に書かれたアプリケーション・プログラムには，変更の必要はない。

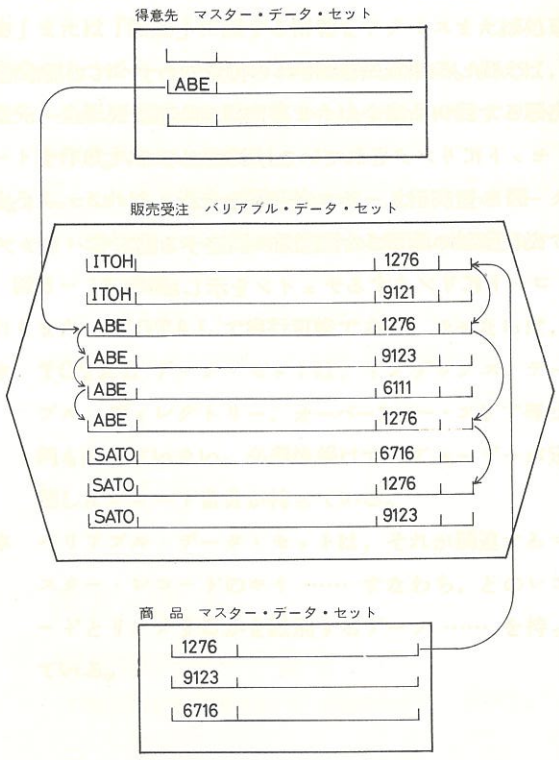


図3－18 得意先―販売受注―商品

これまでに，TOTAL ネットワーク・データ・ベースの関係のうち，下記のタイプのものを紹介してきた。

＊ 1つのバリアブル・データ・セットと1つのマスター・データ・セットの連係。

＊ 複数のバリアブル・データ・セットと1つのマスター・データ・セットの連係。

＊ 1つのバリアブル・データ・セットと複数のマスター・データ・セットの連係。

図3－19 は，さらにもう1つ別のタイプの関係を示している。すなわち，1つのマスター・データ・セットと1つのバリアブル・データ・セットとの間の複数の連係関係である。

この図にある，各販売受注レコードには，2つの日付が含まれている。すなわち，

＊ 注文を受けた日付

＊ 商品を出荷すべき日付　　である。

この販売受注データ・セットに対し2つの連係を持つ日付マスター・データ・セットを作ることにより，以下のような質問に答えることができる。

＊ ある特定年月日，例えば昭和49年1月7日にどんな注文を受けたか。

＊ ある特定年月日，例えば昭和49年1月9日にどの注文に対して出荷すべきか。

また，ある商品の在庫レベルが発注点を下回った時は，コンピュータが自動的に報告を出すようプログラミングをしておくことにより，例外管理および意志決定のルーチン化を進めることもできる。それには，販売受注と商品データ・セットがあればよい。

さらに，部品および資材に関して，部品マスター・データ・セットと資材明細データ・セットを作り，各商品を製造するのにどんな部品を必要とするかを示すこともできる。そうすると，商品の在庫レベルが発注点を下回った時はいつも，必要な資材要求書を自動的に印刷するようにコンピュータ・プログラムを組んでおくことができる。

もう一歩拡張すると，商品の在庫レベルが発注点を下回るといつも，適切な購買指図書と製造指図書をコンピュータが自動的に印刷するよう，プログラムしておくこともできる。図3－20 参照。

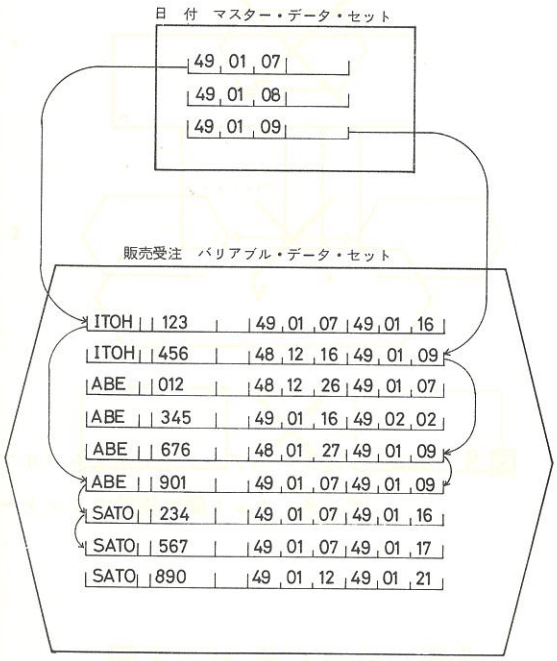


図3－19 日付―販売受注

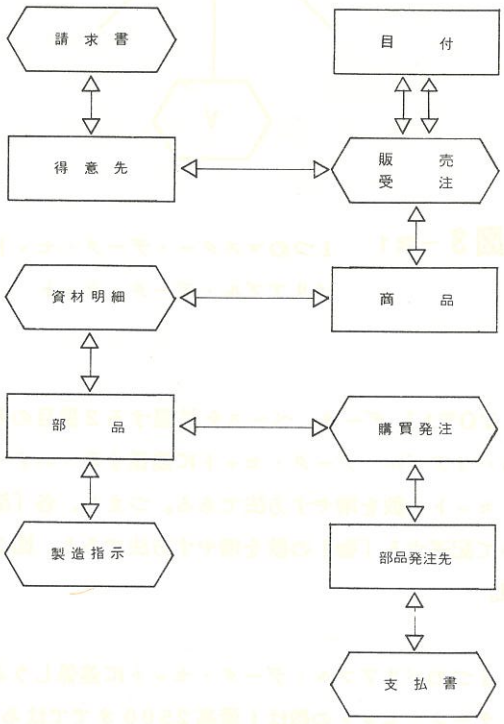


図3－20 発展したデータ・ベース

データ・ベースの拡張

既に理解されたように，TOTAL データ・ベースは，2つのデータ・セットで構成する単純なデータ・ベースから，多数のデータ・セットを含む，大規模かつ複雑なデータ・ベースまで，モジュール的に拡張していくことができる。

データ・ベースを拡張する1つの方法は，1つのマスター・データ・セットに連係するバリアブル・データ・セットの数を増加していく方法である。図3－21 参照。これは得意先，商品，セールスマン等の活動についての記述を増やすことに相当する。

1つのマスター・データ・セットに連係しうるバリアブル・データ・セットの数は（最高2500 までという制限はあるものの），実質的に無限といってよい。

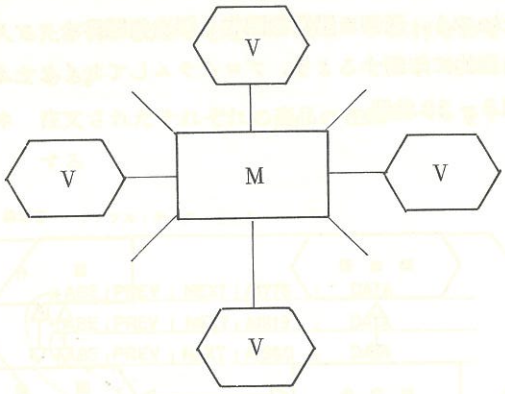


図3－21 1つのマスター・データ・セットに複数のバリアブル・データ・セット

TOTAL データ・ベースを拡張する2番目の方法は，あるバリアブル・データ・セットに連係するマスター・データ・セットの数を増やす方法である。つまり，各「活動」について記述する「物」の数を増やす方法である。図3－22 参照。

1つのバリアブル・データ・セットに連係しうるマスター・データ・セットの数は（最高2500 までではあるが），ここでも無限であるといえる。

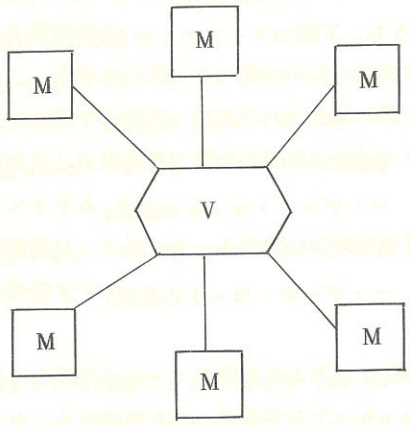


図3－22 1つのバリアブル・データ・セットに複数のマスター・データ・セット

TOTAL データ・ベースを拡張する最後の方法は，マスター・データ・セットとバリアブル・データ・セット間の関係，すなわちリンケージ・パスの数を増やすことである（最高2500まで可能である）。図3－23 参照。

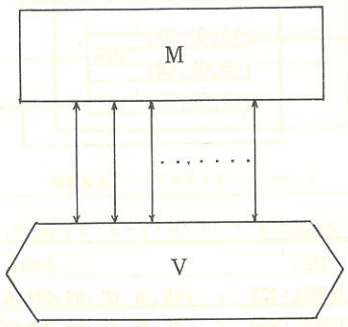


図3－23 マスター・データ・セットとバリアブル・データ・セット間に複数のリンケージ・パス

許されない構造

図3－24は，TOTAL のマスター・データ・セット，バリアブル・データ・セットとして許されない3つの型を示している。

1. まずTOTALは，"独立した"バリアブル・データ・セットを許さない。その理由は，バリアブル・データ・セットの中のあるレコードにアクセスする場合，TOTALはポインター（すなわち，リンケージ・パス）を使い，それは，当該バリアブル・レコードに関係するマスター・レコードが持っている。TOTALは，マスター・レコードのキイを使って …… ランダム化ルーチンを経由して … 当該マスター・レコードをアクセスし，そのマスター・レコードのポインター（リンケージ・パス）を使ってバリアブル・レコードの所在を知る。これは，きわめて当然な順序で，"活動"のレコードをアクセスしようとする時に，その活動を起した"物"と無関係ということはあり得ないのである。どの得意先からの注文か，どの商品に対する注文か判らない受注レコードに意味があるだろうか。

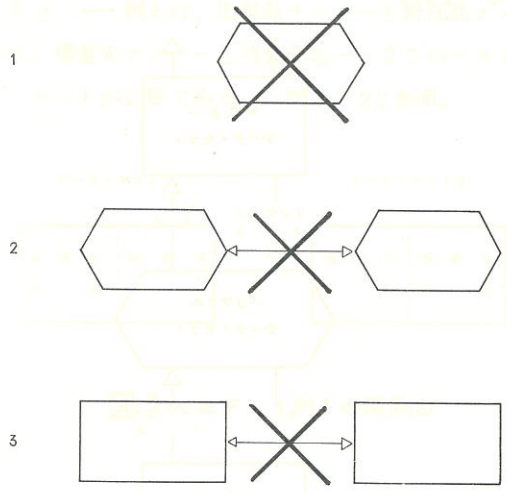


図3－24 許されない構造

2. 次にTOTALは，2つのバリアブル・データ・セットを直接リンクすることを許さない。2つのバリアブル・データ・セットの関係の間には常にマスター・データ・セットが介在する。例えば，受注バリアブル・データ・セットと，請求書バリアブル・データ・セットを直接リンクすることはできない。この場合には，間に得意先マスター・データ・セットが介在するはずであり，それが自然である。得意先Aがある注文をくれたので，我々はAに請求書を送るのである。
3. 最後にTOTALは，2つのマスター・データ・セットを直接リンクすることを許さない。その理由はそのような必要があり得ないからである。もし；マスター・データ・セットAとマスター・データ・セットBをリンクする必要が生じた場合は，マスター・データ・セットAの中にデータとしてBのキイを入れ，Aのキイをの中にデータとして入れるだけでよい。そしてAからBのレコードをアクセスする場合は，Aレコードの中のデータ部にあるBのキイをアクセスし，TOTALのランダム化ルーチンに渡し，Bレコードの所在番地を知ることができる。
つまりTOTALのリンケージ・パスは常にマスター・レコードを，1つまたはそれ以上のバリアブル・レコードに連係するものである。

リンケージ・パスとキイ

どんなマスター・データ・セットとバリアブル・データ・セットの関係も，両面通行的な関係である点を十分理解されたい。図３－25 図参照。つまり，

＊ マスター・レコードは，それにリンクする各バリアブル・チェインの最初と最後を示す「リンケージ・パス」を含んでいる

＊ バリアブル・レコードは，それがリンクする各マスター・レコードの「キイ」を含んでいる

従って，ユーザーはどんなバリアブル・レコードでも，それがリンクしているマスター・レコードから見つけ出すことができるし，また，どんなマスター・レコードでも，それにリンクしたバリアブル・レコードから見つけることができるのである。

それによって，マスター・データ・セットとバリアブル・データ・セットをリンクして，無限に長いストリングや無限に多い階層を作ることもできるわけである。

マスター・データ・セットとバリアブル・データ・セットをこの方法で結ぶことにより，TOTAL データ・ベースは，高度にモジュール化されたものとなっている。例えば，

＊ バリアブル・データ・セットが再ロードされると，そのレコードの位置（すなわちアドレス）も変る。従って，それにリンクしたすべてのマスター・データ・セットのリンケージ・パスを更新しなければならない。しかし，マスター・データ・セットを再ロードしたときは，そのキイは変更されないので他のデータ・セットは全く影響を受けない。

それ故，TOTAL バリアブル・データ・セットが変更されたり（例えばレコード長を長くした場合），損なわれたり（例えば物理的に損傷してしまったとき）したような場合には，それに直接リンクしているマスター・データ・セットのみが影響を受ける。マスター・データ・セットが変更されたり，損なわれても，他のデータ・セットには影響しない。

＊ バリアブル・データ・セットに含まれるマスター・データ・セットへのキイは意味のあるデータである。このキイが存在することにより，マスター・データ・セットを参照する（アクセスする）必要度合が減り，また多くのアプリケーションにおいてマスター・データ・セットからは完全に独立して，バリアブル・データ・セットだけで処理を行うことができる。例えば，この章の始めに説明した販売受注バリアブル・データ・セットは，単にリンクのためのポインターとしてではなく，得意先名，商品名，受注日付，出荷日付，という情報をキイとして持っている。そこでシークェンシャルな売上げリストを作成するプログラムの場合，得意先についてはその名前だけ，商品については商品番号だけ書き出せばよいのであれば，このプログラムを実行するとき，得意先マスターや商品マスターを呼び出すまでもなく，この販売受注バリアブル・データ・セットだけをシークェンシャルに処理すればよい。しかし，もしポインターだけしか持っていなければ，このような売上リストを作るためにも，得意先，商品，日付の各マスター・データ・セットをすべてアクセスしなければならないのである。

データ構造の分類

ネットワーク構造は，データ構造としては最も一般的なものである。従って，データ間のどんな種類の関係であってもTOTAL データ・ベースに収容することができる。データ間の関係には，以下の６種類がある。

＊ １対１

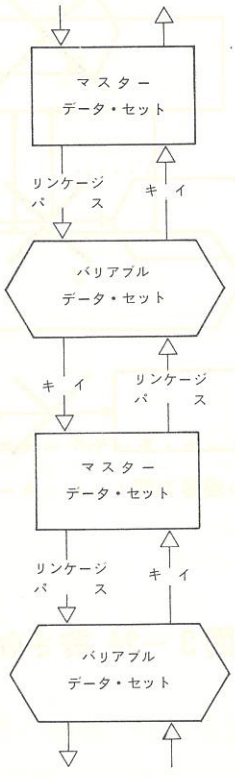


図３－25 リンケージ・パスとキイの役割

＊　1対複数
＊　階　層
＊　複数対複数
＊　逆階層（複数対1）
＊　階層ネットワーク

これらの6つのタイプについて具体的に例をあげて説明する。

1 対 1

「1対1」のデータ関係は，単一のマスター・データ・セットの形であらわすことができる。その最も単純な形の例は，1つのキイとそれに関連するデータをエレメントにもつマスター・データ・セットである。　図3－26参照。

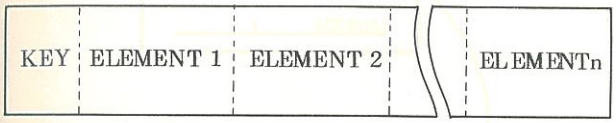


図3－26　1対1の関係(1)

もし，2つのキイでデータをアクセスしなければならないとすると，── 例えば，旧部品ナンバーと新部品ナンバー，または，得意先ナンバーと得意先名── 2つのマスター・データ・セットが必要である。　図3－27参照。

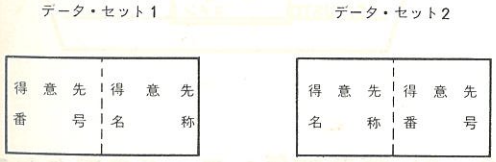


図3－27　1対1の関係(2)

1対複数

「1対複数」のデータ関係は，1つのマスター・データ・セットを1ないし複数のバリアブル・データ・セットへリンクした形で示される。図3－28参照。

もし，その関係が，同じ種類の複数レコードに関連したもの── 例えば，得意先とその得意先からの複数の注文── であるならば，1つのマスター・データ・セットが1つのバリアブル・データ・セットにリンクする形になる。

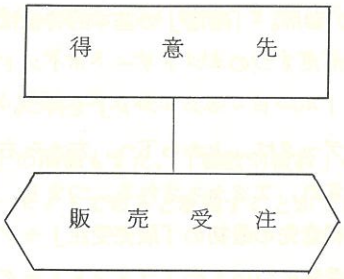


図3－28　1対複数の関係(1)

この関係が，数種類の複数レコードに関係したもの── 例えば，得意先とその得意先からの注文，請求書，入金── である場合には，1つのマスター・データ・セットがいくつかのバリアブル・データ・セットにリンクする形になる。図3－29参照。

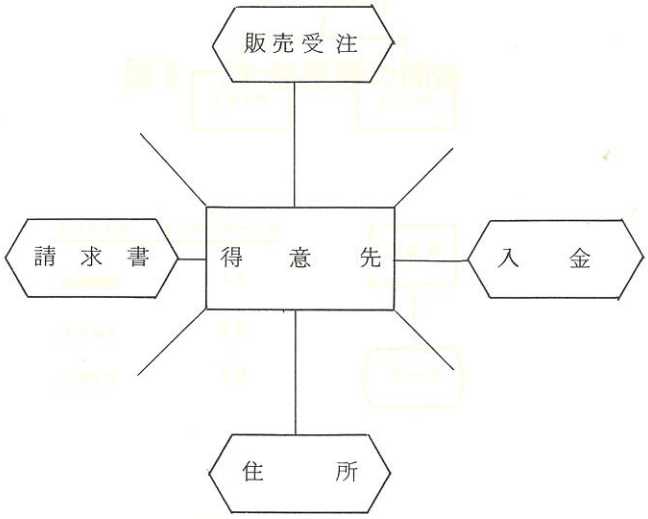


図3－29　1対複数の関係(2)

階　層

「階層」は「1対複数」のデータ関係の特別な型である。図3－30 参照。「階層」の基本的特徴は，

＊ ただ1つのエントリー・ポイントを持つ。すなわち，「ルート・セグメント」を持つ。

＊ データは，上から下へ，左から右への順序でストアされ，アクセスされる。つまり，図の例でいうと，得意先の最初の「販売受注」セグメントの次にその受注についての「コメント」セグメントが来て，さらに，その次にすべての「アイテム」セグメントが来る。それが終わると，2番目の「販売受注」セグメントが来て，その後に，すべてのその「コメント」セグメントと「アイテム」セグメントがストアされる。

TOTAL を用いた「階層」関係を示す最も単純な方法は，「ルート・セグメント」に対して1つのマスター・データ・セットを用い，「従属セグメント」に対して1つのバリアブル・データ・セットを用いることである。バリアブル・データ・セットは複数のレコード・フォーマットを持つことが出来るから，従属セグメントの各タイプに応じて，異ったフォーマットを用いることができる。例えば，「販売受注」セグメントに対してはフォーマット「01」，「コメント」セグメントに対してはフォーマット「02」，「アイテム」セグメントに対してはフォーマット「03」等である。

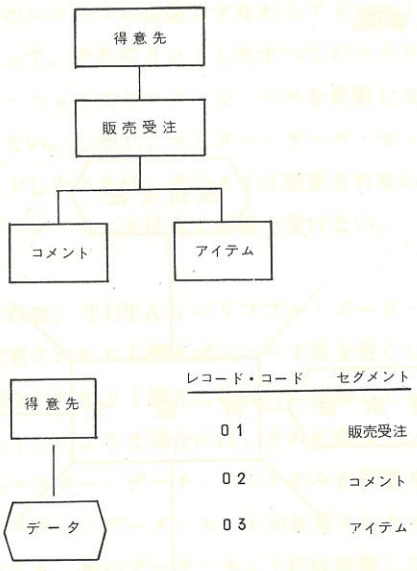


図3－30 階層関係

次の図3－31 は，この階層関係のレコードがアクセスされる順序――すなわち，上から下へ，左から右へ――を示している。

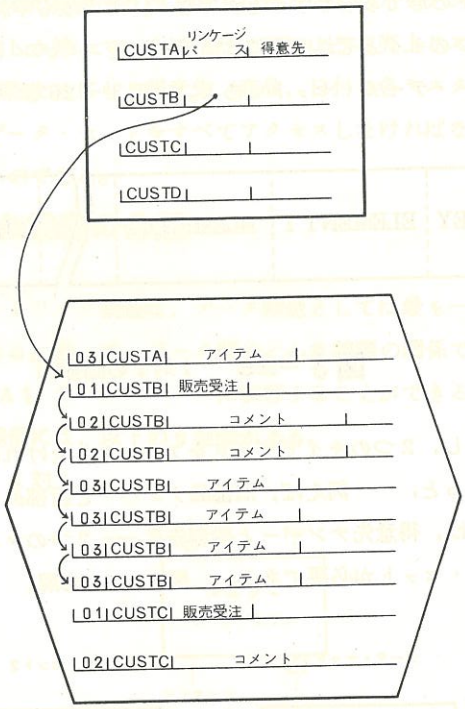


図3－31 階層関係レコードのアクセス順序

複数対複数

「複数対複数」のデータ関係は，いくつかのマスター・データ・セットを1つのバリアブル・データ・セットへリンクした形で示される。図3－32 参照。

例えば，「販売受注」を次のようなマスター・データ・セットへ結びつける。

＊ 受注番号
＊ 得意先
＊ 商品
＊ 受注月日
＊ 出荷月日

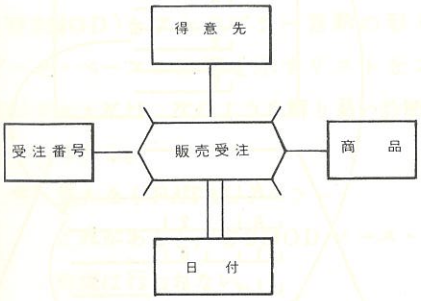


図3－32 複数対複数の関係

逆階層

「逆階層」は，「複数対複数」タイプのデータ関係の特殊ケースに過ぎない。図3－33 参照。また，データ項目を持たないマスター・データ・セットは，バリアブル・データ・セットの単なる索引と考えることができる。

「複数対1」の関係もまた，「複数対複数」のデータ関係の1つのサブ・セットであると看做すことができる。

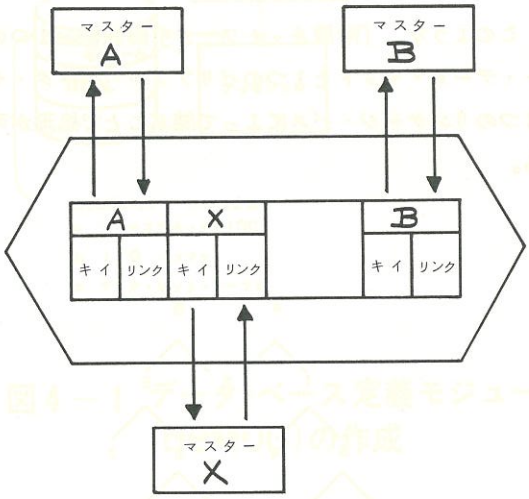


図3－33 逆階層の関係

階層ネットワーク

データ関係の6番目のタイプは，「階層ネットワーク」である。図3－34 参照。これは，「子」が複数の「親」を持つという点で，「単純な階層」とは異っている。

「階層ネットワーク」を上から下へ，もしくは下から上へ処理する事がしばしば必要となる。例えば，ある特定の製品 ── 「A」または「B」 ──に使用するすべての部品を知るばかりでなく，特定の部品 ── 「2」または「7」 ── を使用するすべての製品，半製品を知ることが必要であるような場合である。

このような，「階層ネットワーク」関係は，1つのマスター・データ・セットと1つのバリアブル・データ・セットを2つのリンケージ・パスによって結ぶことで処理が可能となる。

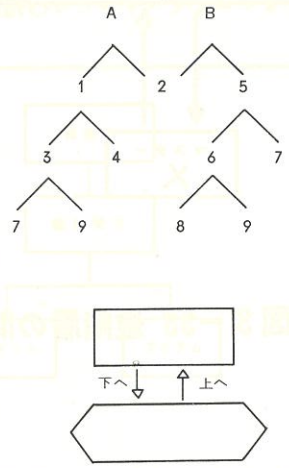


図3－34 階層ネットワークの関係(1)

マスター・データ・セットは，製品とか部品についてのレコードを持つ(すなわち親あるいは子にあたるレコードである)。バリアブル・データ・セットは，"親と子"の関係を示すレコードを持つ。最初のリンケージ・パスは，製品または部品マスター・レコードがそれらを"親"としてもつすべてのバリアブル・レコードへリンクする。そしてこのリンケージ・パスは，製品または部品の"子"をアクセスするのに用いられる。

もう1つのリンケージ・パスは，製品または部品マスター・レコードがそれらを"子"としてもつすべてのバリアブル・レコードへリンクする。このリンケージ・パスは，製品または部品の"親"をアクセスするのに用いられる。図3－35参照。

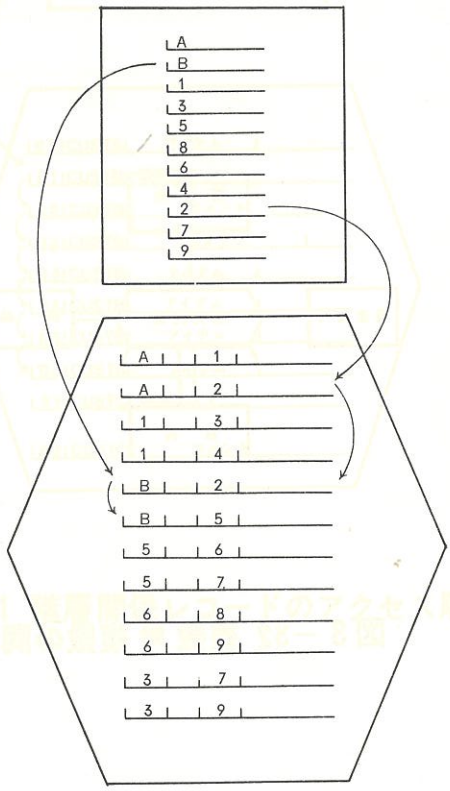


図3－35 階層ネットワークの関係(2)

第4章　データ・ベースの定義

TOTALのデータ・ベースを作るための最初のステップは，データ・ベースを定義することである。それはTOTALのデータ・ベース定義言語（DBDL）を使って行う。

この章ではDBDL　を紹介し，TOTALにデータ・ベースをどのように定義するかを説明する。

データ・ベース定義 モジュール（DBMOD）の作成

TOTALにデータ・ベースを定義するためには，次のステップを必要とする。図4－1参照。

1. データ・ベース定義言語ステートメント（DBDL　ステートメント）をコーディングする。
2. DBGENプログラムを実行する。DBGENはDBDLステートメントを読み，データ・ベース定義モジュール（DBMOD）をアセンブラー言語の形で出力して，データ・ベースの内容を示すリストをプリントする。このリストには，次のような解り易い診断メッセージがリストされる。
   * 重大な（FATAL）エラー
     これがあると，DBMOD ソース・モジュールの作成は行われない。
   * 注意（NOTES）
     エラーとなり得るか，またはDBGENでたてた仮定がリストされる。データ・ベース管理者（DBA）は常に注意深くこれらの"注意"を見なくてはいけない。
3. DBMOD のソース・モジュールをアセンブルする。
4. ステップ3から出たオブジェクト・モジュールをシステム・ライブラリにカタログして，アプリケーション・プログラムがいつでもその定義されたデータ・ベースをアクセスできるようにする。

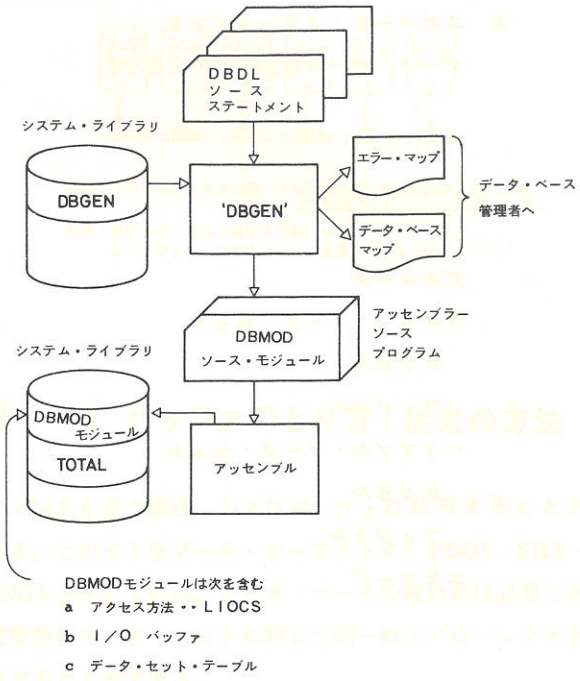


図4－1 データ・ベース定義モジュール（DBMOD）の作成

データ・ベース定義言語（DBDL）

TOTALのデータ・ベース定義言語は，データ・ベース管理者（DBA）が使用する言語である。TOTALのデータ・ベース定義言語（DBDL）は，データ・ベースまたはデータ・ベースのサブ・セットを定義するために1つの定められた形，順を追ってキイ・ワードを指定するようになっている。

DBDL　ステートメントは4つのカテゴリーからなる；

1. プロローグ・ステートメント
   データ・ベースの全般的な特性を定義する。
2. マスター・データ・セット・ステートメント
   各マスター・データ・セットは論理的および物理的に定義される。論理的ステートメントはマスター・データ・セット・レコードのフォーマットを定義する。物理的ステートメントはデータ・セットの物理的特性を定義す

る。

3. バリアブル・データ・セット・ステートメント

各バリアブル・データ・セットも論理的および物理的に定義される。

4. エピローグ・ステートメント

データ・ベース定義の終りを示す。

データ・ベースの定義

プロローグ
マスター・データ・セット
ロジカル
フィジカル
バリアブル・データ・セット
ロジカル
フィジカル
エピローグ

すべてのDBDL ステートメント共通に適用されるシンタックス規則は次の通りである。

* すべてのステートメントはカラム1より始めなければいけない。カラム1がブランクで始まるステートメントはすべてコメントとして扱われる。
* ステートメントはブランクがあるとそこで終わりを示す。それ以降のエントリーはすべてコメントとみなされる。
* 大文字のエントリーおよび句読点は指定の通りに書かなければならない。
* "mmmm"にはマスター・データ・セット名を記入する。
* "vvvv"にはバリアブル・データ・セット名を記入する。
* "xxxx"には任意の文字(A～Z, 0～9, #, $および@)を記入する。
* "n"は任意の数字を記入する。
* "dddd"にはDASD 装置(例えば3330,2314, 2311)の名前を記入する。

すべてのマスター・データ・セットは, バリアブル・データ・セットの定義に先だち定義されなければならない。

プロローグ・ステートメント

DBDL のプロローグ・ステートメントは次のとおりである。

プロローグ・ステートメント

```
BEGIN-DATA-BASE-GENERATION:
DATA-BASE-NAME = xxxxx
SHARE-IO:
IOAREA = xxxx = n
.
.
.
IOAREA = xxxx = n
BLOCK-HOLD = YES
```

BEGIN-DATA-BASE-GENERATION:

データ・ベース定義の最初のステートメントである。

DATA-BASE-NAME = xxxxxx

データ・ベースに付ける名前を6桁の英数字で指定する。

SHARE-IO:

IOAREA = xxxx = n

次の節で説明する。

BLOCK-HOLD = YES

このステートメントはTOTAL 4 の場合は複数のプログラムが走る「マルチ・タスク」のとき, データ・ベースの同時更新を防ぐために必要な宣言である。このステートメントによって, オペレーティング・システムのENQUEUE／DEQUEUE機能を呼び, 2つ以上のタスクが同じレコードを同時に更新することを防ぐ。すなわち, 1つのタスクに1データ・セット当り1レコードを占有(HOLD)させ, そのレコードの更新が終了するかそのタスクが同じデータ・セットから別のレコードを読む命令を出す迄, 占有を続けさせる。

このステートメントはTOTAL 5／6では必要としない。TOTAL 5／6は, 同じレコードを2つ以上のタスクが同時更新しようとするときは, 自分の"タスク・ス

ケジュール機能（第８章参照）"を作動させる。

I／O　バッファの共有

従来から，パフォーマンスとエフィシェンシーとはお互に両立していないといわれている。つまり，どちらか一方を犠牲にすることによって，はじめてもう一方が達成できた。高いパフォーマンスをあげると，すなわちトランザクションの更新を迅速に行い，データへ素早くアクセスを行うことは，データ・ベース・システムにとって一番重要なことであり，そのためにデータ・ベースへの移行に伴いしばしばコンピュータ設備の面で，莫大な拡張が必要とされた。より大型の機械を，より大きなコア容量を，またはより多くのDASDスペースを求めることになる。

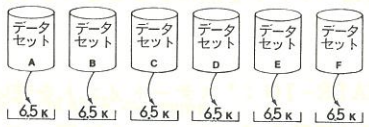
しかしながら，このことはTOTALにはあてはまらない。TOTALの最も重要な利点の１つは，パフォーマンスもエフィシェンシーも，ともに最適化されるということである。このことは次のような実例を参照願いたい。

ある大手自動車メーカーがTOTALを使って全販売情報のシステムを実施したが，この会社の場合，５００,０００件以上のトランザクションを３０スピンドル３３３０データ・ベースによって毎日処理している。

ある大手鉄道会社がTOTALを使って運賃および車両の管理システムをはじめたが，毎日2,000,000件以上のトランザクションを処理している。このシステムでは毎秒７０件のトランザクションを処理することが可能である。

アメリカにある，ある日本の子会社が注文処理，在庫管理および得意先勘定処理を行うためにTOTALで総合的なデータ・ベース・システムを実施した。このシステムは，９６KのIBM 360／30で動いているオンライン・システムである。

前述の通りTOTALの必要とするコア容量はそれほど多くない。TOTALがいかにパフォーマンスとエフィシェンシーの両方を最適化しているかは，Ｉ／Ｏバッファリングに見ることができる。ある１つのデータ・ベース・システムの下では，通常多くのデータ・セットが同時に処理される。この場合高いパフォーマンスは，それぞれのデータ・セットが大きなバッファを持つことによって達成することができる。しかし，ここに示してある図４－２の例では，このアプローチによると非常に大量のコア容量を必要としていることがわかる。コアの効率は，高いパフォーマンスを達成するために犠牲にされたことになる。



条件：３３３０ディスク使用、ブロック・サイズは½トラック

結果：各データ・セット当り１つのバッファ
６バッファ　＊　6.5Kバイト＝３９Kバイト

図４－２　バッファのとり方：従来の方法

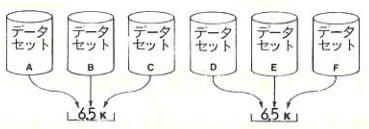
TOTALを使う場合，Ｉ／Ｏバッファは共有することができる。このことをプール・シェアリング（POOL SHARING）と呼んでいる。データ・ベースの設計者は必要とあらば複数のデータ・セットに対して同一のＩ／Ｏバッファを割当てることができる。

図４－３に示してある通り，同一のバッファに対して１つ以上のデータ・セットを割当てることによって次のことが可能である。

a　コアを効率よく活用する。

b　ディスクの効率には変りはない。

c　スループットはほとんど，または全く低下することはない。もちろん，データ・ベースの設計者は，プール・シェアリングを使用しても，複数のデータ・ベースが１



条件：３３３０ディスク使用、ブロック・サイズは½トラック

結果：１つのバッファを３つのデータ・セットで共有すれば
２バッファ　＊　6.5Kバイト＝１３Kバイト
コアの節約　２６K

図４－３　バッファのとり方：TOTALの方法

つのＩ／Ｏバッファに対して取り合いを起したり，又そのためにパーフォーマンスが悪くなるようなことが起きないことを確かめておく責任がある。そして，これを防ぐには，同時にアクセスする可能性があるファイルに対してはバッファを別にすることである。

"SHAIR-IO："ステートメントを使い，データ・ベース設計者は複数のデータ・セットに同一のＩ／Ｏバッファを割当てることができる。共有される各Ｉ／Ｏバッファは"IOAREA＝xxxx＝n"ステートメントにより定義される。Ｉ／Ｏバッファの名称"xxxx"は任意の4桁の英数字である。

TOTAL 5／6ではパラメーター"n"で"数"を指定することにより，Ｉ／Ｏバッファを指定した数だけ作ることができる。

Ｉ／Ｏバッファをどの程度まで共有できるか，あるいはどのデータ・セットはＩ／Ｏバッファを共有してもよいかを決めるに際して，重要なことがある。それは，もし要求されたレコードが既にどこかのバッファ内にあれば，TOTAL はこのレコードをディスクから読まないということである。　図4－4参照。

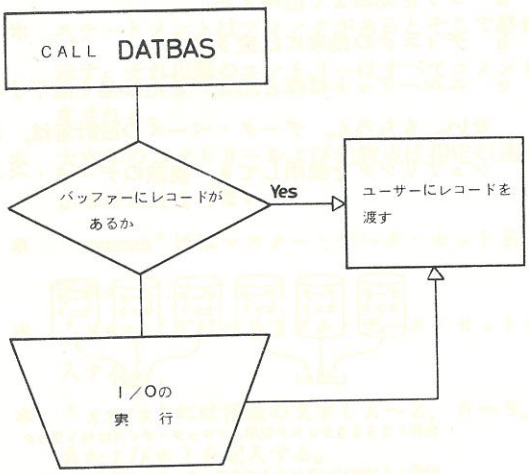


図4－4　バッファ・ロジック

マスター・データ・セット　ステートメント

標準フォーマット

先に述べたように，マスター・データ・セット・レコードは単一のユニークなキイでアクセスされる。そして同一のマスター・データ・セットにあるすべてのレコードは同じ長さとフォーマットをもつ。図4－5はマスター・データ・セット・レコードの標準フォーマットを示す。

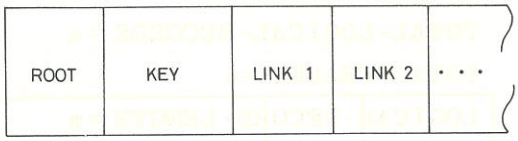
レコードの構成要素は，

ルート：　マスター・データ・セット・レコードの最初の8バイトのことである。これはマスターの「シノニム（SYNONYM）」レコードと「ホーム」レコードを関係づけるためにTOTALが使うフィールドである。ホーム・レコードのルートはシノニムチェインの最初と最後のレコードのポインター（すなわち，リラティブ・レコード・ナンバー（RRN））をもっている。各々のシノニム・レコードのルートは，当該シノニム・レコード・チェインの直前直後のレコードを指している。

キイ：　マスター・レコードを指すユニークな識別名である。キイはレコードをストアしたり，アクセスするときに使われる。各々のマスター・レコードはキイをもたねばならないし，しかもたゞ1つに限られる。値（VALUE）に対してそれぞれ1つのマスター・レコードが存在する。

リンク　1：
リンク　2
・
・
・　マスター・レコードをバリアブル・レコードに関連づけるためのリンケージ・パスである。マスター・レコードはバリアブル・データ・セットとの1つの関係ごとに1つのリンケージ・パスをもつ。8バイトの長さで，チェイン内の最初と最後のバリアブル・レコードのRRN をポインターとして格納している。もしマスター・データ・セットが独立であればリンクするバリアブル・ファイルがないのであるから，リンケージ・パスはない。

エレメント 1： マスター・レコード内のデータ・エレメント（もしあれば）の名前，長さを指定する。
エレメント 2
・
・
・
エレメント N



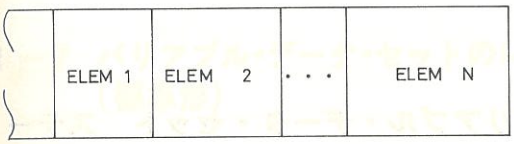


図4－5 マスター・データ・セットのレコード

DBDL ステートメント

マスター・データ・セットの論理的特性を定義するDBDLステートメントは次のとおりである。

マスター・データ・セット・ステートメント

```
BEGIN-MASTER-DATA-SET：
DATA-SET-NAME＝mmmm
IOAREA＝xxxx
MASTER-DATA：
mmmmROOT＝8
mmmmCTRL＝n
mmmmLKxx＝8＝vvvv
・
・
・
mmmmLKxx＝8＝vvvv
mmmmxxxx＝n
・
・
・
mmmmxxxx＝n
END-DATA：
```

DATA-SET-NAME＝mmmm

各マスター・データ・セットはユニークな4桁の英数字名"mmmm"をもつ。この"mmmm"の名前はこのデータ・セットの各エレメント名の頭4文字としても使われる。

IOAREA＝xxxx

このマスター・データ・セットがプロローグで定義したI／Oバッファの1つを共有する場合に使う。もし，このマスター・データ・セットが専有の（すなわち共有しない）I／Oバッファをもつ場合には，このステートメントは必要ない。

mmmmROOT＝8

マスター・レコードの最初の8バイトを定義する。TOTALはこのエリアをシノニムの処理のために使う。

mmmmCTRL＝n

このマスター・データ・セットのユニークなコントロール・キイを定義する。"n"はキイの長さを示し1～256バイトの範囲である。

mmmmLKxx＝8＝vvvv

このマスター・データ・セットから，"vvvv"という名前のバリアブル・データ・セットへのリンケージ・パスを定義する。LKxx はパスの番号で1つのマスター・データ・セットは最高2500迄リンケージ・パスをもつことができる。各々は8バイトの長さである。

mmmmxxxx＝n

エレメントの名前と長さを定義する。1つのマスター・データ・セットがもつエレメントの数に制限はない。

前述したように，エレメントはTOTALがデータをアクセスする際の最小の単位である。それ故にエレメント内をアイテムごとにさらに定義する必要はない。

しかし良いドキュメンテーションを作るための習慣としてアイテムやその説明を定義しておくことを勧める。各アイテ

ムにコメント・カード（カラム１にブランク）を用いるのも良い方法である。このコメント・カードはエレメントの定義ステートメントの後におく。

またステートメントの右側に１つ以上のブランクをおけば，その右側もコメントとして扱われるので，コントロール・キイ，データ・エレメントあるいは，アイテムの説明 － 例えばリテラルか，バイナリーか，パックト・デシマルか － をステートメントの＂ｎ＂のあとに，１つ以上のブランクをおいて記述しておくのもよい方法である。図４－６参照。

ROOT	得意先番号 CUSTOMER NUMBER	販売受注 バリアブル・データ・セットへリンク	請求書 バリアブル・データ・セットへリンク
8	12	8	8

得意先名称 CUSTOMER NAME		日付 TRANSACTION			得意先住所 CUSTOMER ADDRESS
LAST	FIRST	年	月	日	
20	20	2	2	2	30

```
BEGIN-MASTER-DATA-SET:
DATA-SET-NAME=CUST
IOAREA=MAS1
MASTER-DATA:
CUSTROOT=8
CUSTCTRL=12          CUSTOMER NUMBER
CUSTLKO1=8=CORD      LINK TO CUSTOMER ORDER
CUSTLKO2=8=INVC      LINK TO INVOICE
CUSTNAME=40
 CUSTLNAM=(20)  ALPHA(LAST)
 CUSTFNAM=(20)  ALPHA(FIRST)
CUSTDATE=6
 CUSTYEAR=(2)   NUMERIC
 CUSTMNTH=(2)   NUMERIC
 CUSTDAY=(2)    NUMERIC
CUSTADDR=30     ALPHA
 .
 .
 .
```

図４－６　例 DBDLにコメントを付ける

マスター・データ・セットの物理的特性を定義するDBDLステートメントは次のとおりである。

マスター・データ・セット・ステートメント

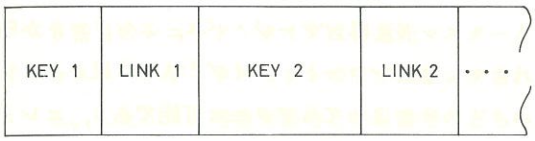
```
DEVICE＝dddd
TOTAL-LOGICAL-RECORDS＝n
TOTAL-TRACKS＝n
LOGICAL-RECORD-LENGTH＝n
LOGICAL-RECORDS-PER-BLOCK＝n
LOGICAL-BLOCKS-PER-TRACK＝n
END-MASTER-DATA-SET：
```

バリアブル・データ・セット　ステートメント

標準フォーマット

図４－７はバリアブル・データ・セット・レコードの標準フォーマットを示す。レコードの構成要素は，

キイ　１	このレコードがマスター・データ・セットとの間に持つ関係（すなわちリンケージ・パス）ごとに，１つのキイと１つのリンクが必要である。キイは関係するマスター・データ・レコードのキイと同じである。各リンクは８バイトのリンケージ・パスであり関連しているチェインの直前・直後のバリアブル・レコードのＲＲＮを指している。
リンク　１	
キイ　２	
リンク　２	
・	
・	
・	
エレメント　１	このレコードに含まれるエレメント（データ部）である。
エレメント　２	
・	
・	
・	
エレメント　Ｎ	



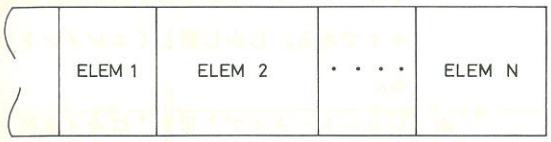


図4－7 バリアブル・データ・セットのレコード（標準形）

DBDL ステートメント

バリアブル・データ・セットの論理的特性を定義するDBDL ステートメントは次のとおりである。

バリアブル・データ・セット・ステートメント

BEGIN-VARIABLE-ENTRY-DATA-SET：
DATA-SET-NAME＝vvvv
IOAREA＝xxxx
BASE-DATA：
{ vvvvxxxx＝n＝mmmm CTRL
{ mmmm LKxx＝8
・
・
・
vvvvxxxx＝n
・
・

vvvvxxxx＝n

DATA-SET-NAME＝vvvv

各バリアブル・データ・セットはユニークな4桁の英数字名"vvvv"をもつ。この"vvvv"の名前はこのデータ・セットの各エレメント名の頭の4文字としても使われる。

IOAREA＝xxxx

このバリアブル・データ・セットがプロローグで定義したI／Oバッファの1つを共有する場合に使う。もしこのバリアブル・データ・セットが専有の（すなわち共有しない）I／Oバッファをもつ場合には，このステートメントは必要ない。

vvvvxxxx＝n＝mmmm CTRL
mmmm LKxx＝8

これらは常に対で使われ，マスター・データ・セット"mmmm"からこのバリアブル・データ・セットへのリンケージ・パスを定義する。1つのバリアブル・データ・セットは最高2500迄のリンケージ・パスをもつことができる。

vvvvxxxx＝n

エレメントの名前と長さを定義する。1つのバリアブル・データ・セットがもつエレメントの数に制限はない。

マスター・データ・セットの場合と同様に，良いドキュメンテーションを作るための習慣として，コメント・カード（カラム1にブランクまたはステートメントの右側に1つ以上のブランク）を利用してアイテムやデータ・タイプなどを定義しておくことを勧める。

コード付レコード

マスター・データ・セットと同様，1つのバリアブル・データ・セットの中のレコードはすべて同じ長さでなければいけない。しかし，1つのバリアブル・データ・セットの中でレコードによって異なる複数のレコード・フォーマットを使用することができる。

もし異なるレコード・フォーマットを使う場合には，各々

のフォーマットをTOTALのDBMODで定義しなくてはいけない。このようなレコードは，そのレコードの最初の2バイトに"レコード・コード"をもたなければいけない。このレコードによりTOTALはレコードのフォーマットを識別することができる。このようなレコードを「コード付レコード」と呼ぶ。

もし，バリアブル・データ・セットの中のすべてのレコードが同じフォーマットであれば，レコード・コードは不必要であり，その2バイトは別のデータのために使うことができる（つまりこの2バイトはレコード・コードとして割り当てる必要はない）。

コード付レコードは次の2つの部分に分かれる。図4－8参照。

ベース・データ部

この部分は，このデータ・セットの中のすべてのレコードに共通なキイ，リンケージ・パス，エレメントより成る。従って図4－8においてすべてのレコードはキイ1またはキイ2でアクセスすることができ，それぞれエレメント1を含んでいる。

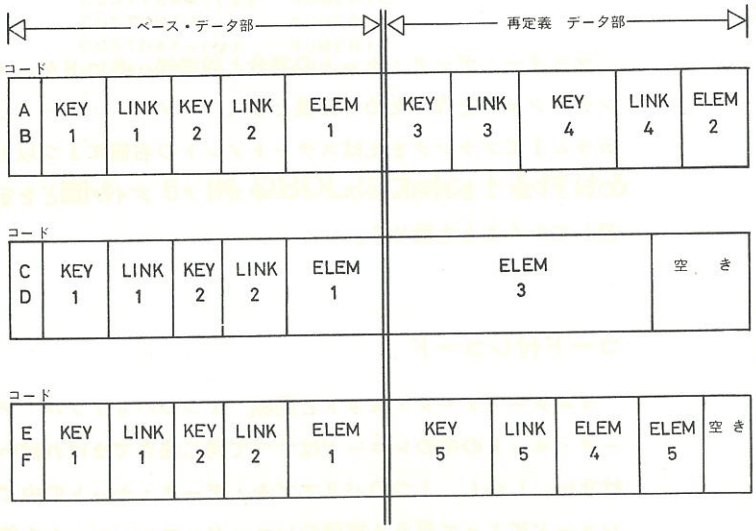


図4－8 コード付レコードのフォーマット

再定義データ部

この部分は，このタイプのレコードにのみ該当する…イ，リンケージ・パス，エレメントから成る。従って…

＊ レコード・タイプ"ＡＢ"はキイ3またはキイ4によってもアクセス可能であり，エレメント2を含んでいる。

＊ レコード・タイプ"ＣＤ"は，ベース・データ部にあるキイによってのみアクセスが可能である。つまり，データ・セットのすべてのレコードに共通のキイである。しかし新しくエレメント3を含んでいる。

＊ レコード・タイプ"ＥＦ"はキイ5によってもアクセスできるし，エレメント4および5を含んでいる

なぜコード付レコードが必要となるか？

ここでバリアブル・データ・セットを別の角度から，つまりレコードの「プール」として考えてみよう。このプールの中で，異ったレコードが異った機能をもっていることは当然考えられることである。リンクのためだけのレコードもあるつまり異ったマスター・データ・セット同志結びつけるためのレコードである。また関連マスター・データ・セットを補足するデータのみをもつレコードもある。異ったレコードは各々異った機能をもつのであるから，これらの異ったレコードがそれぞれ異ったマスター・データ・セットにリンクされる事もあり得る。

図4－9は，マスター・データ・セットに関連する特別の得意先注文についての情報を，コード付レコードで示したものである。

最初のレコードは，得意先マスターと注文マスター・データ・セットをリンクしたものである。したがって，得意先番号または注文番号をキイにしてアクセスできる。アプリケーション・プログラムには，ベース・データ部および再定義データ部も共に利用可能である。

2番目のレコード・タイプは特別の得意先注文の…

メントが含まれている。これは得意先マスターにのみリンクされている。

3番目のレコード・タイプは得意先マスターと品目マスター・データ・セットの2つにリンクされている。したがって得意先番号と品目番号のどちらからでもアクセスすることができる。

この図にはリンケージ・フィールドは示されていないが，これは既に示した図の通り，各々のコントロール・キイの右側に8バイトもっている。

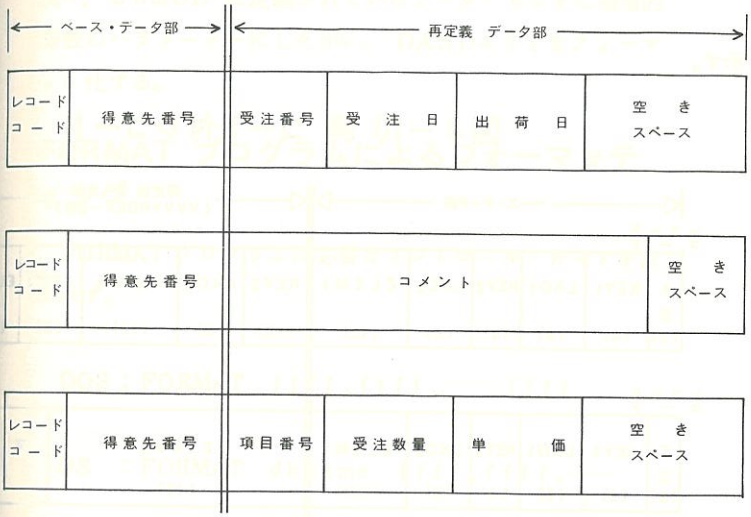


図4－9 例 コード付レコード

コード付レコードを含むバリアブル・データ・セットの論理的特性を定義するDBDL ステートメントは次のとおりである。前述のコードをもたないレコードの定義 DBDL ステートメントとの違いは，ベース・データ部に2つの定義が加わることである。

バリアブル・データ・セット・ステートメント

```
BEGIN-VARIABLE-ENTRY-DATA-SET：
DATA-SET-NAME＝vvvv
IOAREA＝xxxx
BASE-DATA：
vvvv CODE＝2
vvvvxxxx＝n＝mmmm CTRL
mmmm LKxx＝8
      ・
      ・
      ・
vvvvxxxx＝n
      ・
      ・
      ・
vvvv RDEF＝n
RECORD-CODE＝xx
vvvvxxxx＝(n)＝mmmm CTRL
mmmm LKxx＝(8)
      ・
      ・
      ・
vvvvxxxx＝(n)
      ・
      ・
      ・
vvvvxxxx＝(n)
END-DATA：
```

ベース・データ部

再定義データ部

vvvvCODE＝2

レコードの最初の2バイトでそのレコードのフォーマットを識別するためのコードがはいる。

vvvvRDEF＝n

このバリアブル・データ・セット・レコードの再定義データ部の長さ'n'を定義する。このエレメントは，DBCL CALLステートメントで，バリアブル・データ・セットの再定義データの全データをアクセスするために使われる。

レコードのベース・データ部に定義したリンケージ・パスを使えば，当該バリアブル・データ・セットの中の全レコードをアクセスすることができる。

RECORD-CODE＝xx

このバリアブル・データ・セット・レコードの再定義データ部の定義(フォーマット)はこのステートメントから始まる。"xx" のバリューはエレメント

"vvvvCODE" にストアされる。

各RECORD-CODE＝xx ステートメントに続いて，このフォーマット・タイプのレコードの再定義データ部に含まれるコントロール・キイ，リンケージ・パスおよびエレメントを定義する。その次に，次のフォーマット・タイプのレコードの再定義データを定義するRECORD-CODE＝xx が続く。

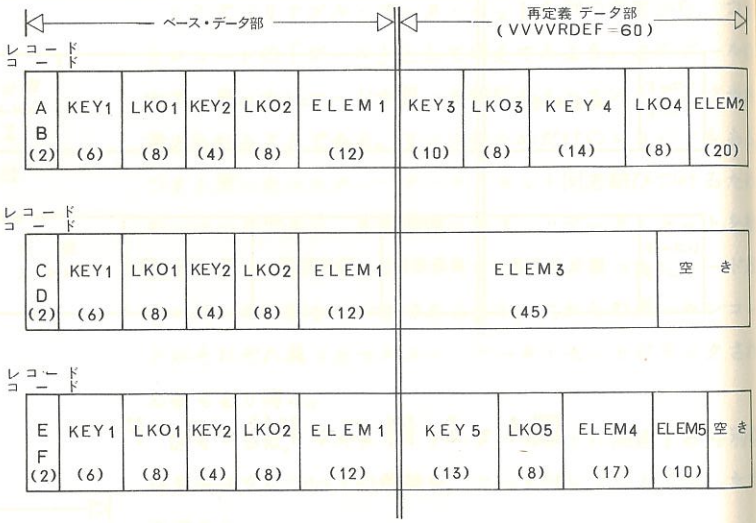
1つのバリアブル・データ・セットが持つレコードのフォーマット・タイプの数（従って，RECORD-CODE＝xx ステートメントの数）に制限はない。

コード付バリアブル・データ・セットの定義例を次に示す。

```
BEGIN-VARIABLE-ENTRY-DATA-SET：
DATA-SET-NAME＝vvvv
IOAREA＝VAR1
BASE-DATA：
vvvv CODE＝2
vvvv KEY1＝6＝mmmm CTRL
mmmm LK01＝8
vvvv KEY2＝4＝mmmm CTRL
mmmm LK02＝8
vvvv ELM1＝12
vvvv RDEF＝60
RECORD-CODE＝AB
vvvv KEY3＝(10)＝mmmm CTRL
mmmm LK03＝(8)
vvvv KEY4＝(14)＝mmmm CTRL
mmmm LK04＝(8)
vvvv ELM2＝(20)
RECORD-CODE＝CD
vvvv ELM3＝(45)
RECORD-CODE＝EF
vvvv KEY5＝(13)＝mmmm CTRL
mmmm LK05＝(8)
vvvv ELM4＝(17)
vvvv ELM5＝(10)
END-DATA：
    •
    •
    •
END-VARIABLE-DATA-SET：
```

この例を図解したのが次の図4－10である。

図4－10 例 コード付レコード



エピローグ・ステートメント

すべてのマスターおよびバリアブル・データ・セットを定義した後に使われる最後の DBDL ステートメントは，次のエピローグ・ステートメントである。

エピローグ・ステートメント

```
END-DATA-BASE-GENERATION：
```

これによってDBGENに DBDL ステートメントの終りを知らせる。

第5章　データ・ベースのフォーマッティング

TOTALデータ・ベースの作成 …… すなわち，レコードをロードまたは追加する …… に先だち，データ・セットが格納されるDASD エリアは，TOTAL FORMAT プログラムによりフォーマット化されなければならない。FORMAT プログラムはフォーマット・コントロール・カードを読み，DBMOD に定義されているデータ・セットの物理的特性のパラメーターにしたがい。DASDエリアをフォーマット化する。

FORMAT プログラムによるフォーマッティング

FORMAT プログラムに必要なコントロール・カードを次に示す。

DOS：FORMAT　ffff,ffff,……,ffff

OS　：FORMAT　dbname　ffff,ffff,……

dbname はDBMOD に定義されている6桁のデータ・ベース名，ffffはフォーマットされるデータ・セット名である。

FORMAT プログラムは，コントロール・カードで指定されたデータ・セットの格納に必要なエリアをブランク・レコードで満し，必要となるすべてのコントロール・レコードを設定して，各データ・セットのエリアをフォーマット化する。図5－1参照。

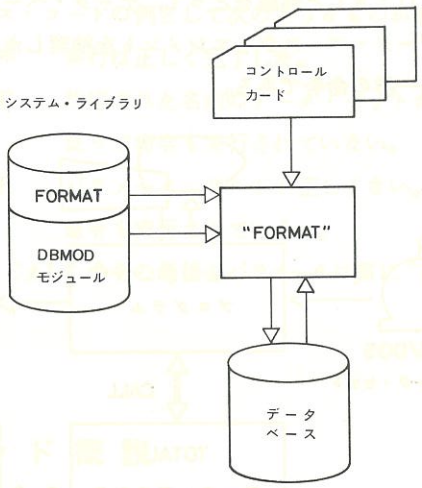


図5－1

データ・ベースのフォーマッティング

"vvvvCODE" にストアされる。

各RECORD-CODE＝xx ステートメントに続いて，このフォーマット・タイプのレコードの再定義データ部に含まれるコントロール・キイ，リンケージ・パスおよびエレメントを定義する。その次に，次のフォーマット・タイプのレコードの再定義データを定義するRECORD-CODE＝xx が続く。

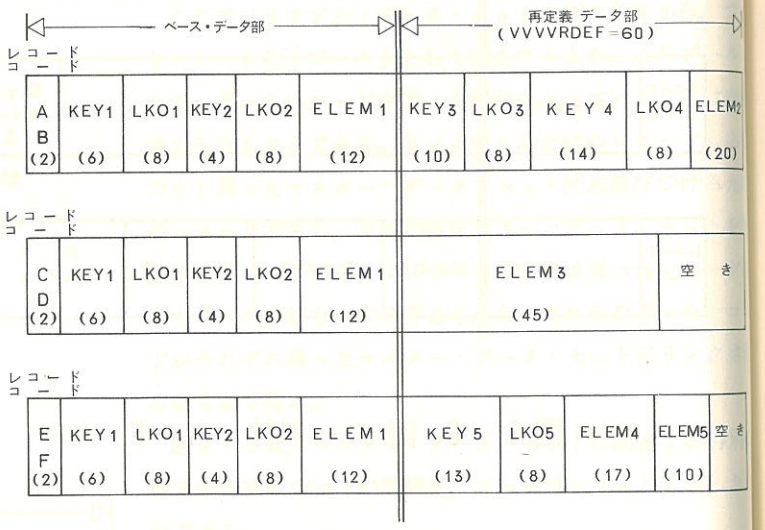
1つのバリアブル・データ・セットが持つレコードのフォーマット・タイプの数（従って，RECORD-CODE＝xx ステートメントの数）に制限はない。

コード付バリアブル・データ・セットの定義例を次に示す。

```
BEGIN-VARIABLE-ENTRY-DATA-SET：
DATA-SET-NAME＝vvvv
IOAREA＝VAR1
BASE-DATA：
vvvv CODE＝2
vvvv KEY1＝6＝mmmm CTRL
mmmm LK01＝8
vvvv KEY2＝4＝mmmm CTRL
mmmm LK02＝8
vvvv ELM1＝12
vvvv RDEF＝60
RECORD-CODE＝AB
vvvv KEY3＝(10)＝mmmm CTRL
mmmm LK03＝(8)
vvvv KEY4＝(14)＝mmmm CTRL
mmmm LK04＝(8)
vvvv ELM2＝(20)
RECORD-CODE＝CD
vvvv ELM3＝(45)
RECORD-CODE＝EF
vvvv KEY5＝(13)＝mmmm CTRL
mmmm LK05＝(8)
vvvv ELM4＝(17)
vvvv ELM5＝(10)
END-DATA：
    •
    •
    •
END-VARIABLE-DATA-SET：
```

この例を図解したのが次の図4－10である。

図4－10 例 コード付レコード



エピローグ・ステートメント

すべてのマスターおよびバリアブル・データ・セットを定義した後に使われる最後の DBDL ステートメントは，次のエピローグ・ステートメントである。

エピローグ・ステートメント

```
END-DATA-BASE-GENERATION：
```

これによってDBGENに DBDL ステートメントの終りを知らせる。

第5章　データ・ベースのフォーマッティング

TOTALデータ・ベースの作成 …… すなわち，レコードをロードまたは追加する …… に先だち，データ・セットが格納されるDASD エリアは，TOTAL FORMAT プログラムによりフォーマット化されなければならない。FORMAT プログラムはフォーマット・コントロール・カードを読み，DBMOD に定義されているデータ・セットの物理的特性のパラメーターにしたがい。DASDエリアをフォーマット化する。

FORMAT プログラムによるフォーマッティング

FORMAT プログラムに必要なコントロール・カードを次に示す。

DOS：FORMAT　ffff,ffff,……,ffff

OS　：FORMAT　dbname　ffff,ffff,……

dbname はDBMOD に定義されている6桁のデータ・ベース名，ffffはフォーマットされるデータ・セット名である。

FORMAT プログラムは，コントロール・カードで指定されたデータ・セットの格納に必要なエリアをブランク・レコードで満し，必要となるすべてのコントロール・レコードを設定して，各データ・セットのエリアをフォーマット化する。図5－1参照。

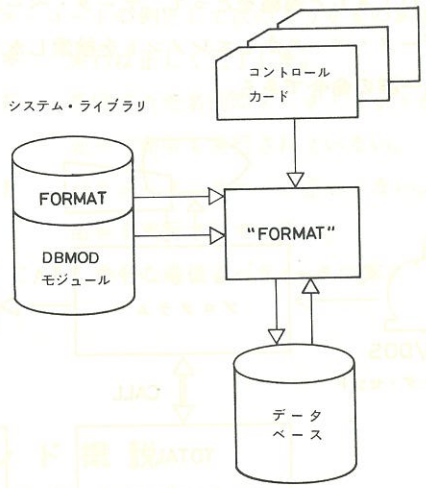


図5－1

データ・ベースのフォーマッティング

第6章　データ・ベースへのアクセス

本章ではＴＯＴＡＬのデータ・ベース・コマンド言語（ＤＢＣＬ）について紹介する。

ＤＢＣＬは，プログラマーがアプリケーション・プログラムで，ＴＯＴＡＬと連絡をとって，データ・ベースを更新したり，データ・ベースからエレメントを検索したりするために用いる一群の命令である。

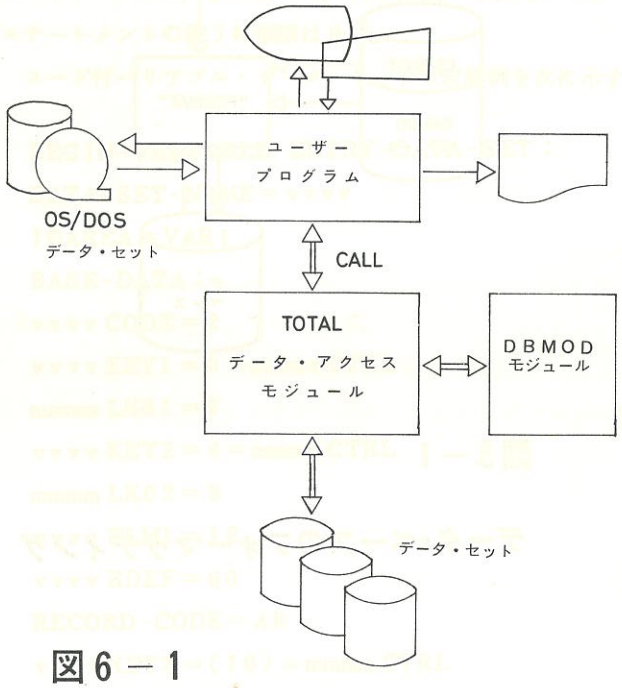


図６－１
データ・アクセス・モジュール

データ・ベース・コマンド言語（ＤＢＣＬ）

ＴＯＴＡＬ　ＤＢＣＬはプログラマーがアプリケーション・プログラムとデータ・ベースと連絡しあうために用いる言語である。ＴＯＴＡＬ　ＤＢＣＬはそれ自体では完全な言語ではなく，プログラム・ロジックやデータを操作するなどの機能のためには，ホスト言語の助けを借りなければならない。

ＴＯＴＡＬ　ＤＢＣＬはコール（ＣＡＬＬ）またはマクロ（ＭＡＣＲＯ）レベルで，ＣＯＢＯＬ，ＦＯＲＴＲＡＮ，ＡＳＩ－ＳＴ，ＰＬ／１，アセンブラーといったホスト言語と組み合せて使われる。従って，ユーザーのアプリケーション・プログラムはホスト言語によるステートメントとＤＢＣＬ命令とが混り合ったものとなる。ＴＯＴＡＬ　ＤＢＣＬはデータ・ベースを処理操作する言語である。図６－１参照。

すなわち，データ・ベースからデータを検索したり，データ・ベースにデータやデータ間の関連を追加，変更，削除をするための　ＣＡＬＬ　の中味はすべてＴＯＴＡＬ　ＤＢＣＬで指定する。ＴＯＴＡＬ　ＤＢＣＬにはデータ操作が正しく行なわれるための広範な保護機能と解析機能が備っている。命令がうまく実行されたか，またはうまくいかなかったときのステイタスはどんな状態であったかを知らせる診断メッセージが用意されている。例えば既にデータ・ベース中に存在するレコードと重複したレコードを追加しようとすると，ＴＯＴＡＬ　ＤＢＣＬが当該レコードが既にデータ・ベース中にある旨を表示してくれる。

ＴＯＴＡＬがアクセスするのはエレメントのレベルである。ＴＯＴＡＬ　ＤＢＣＬ命令を実行すると，その命令中のエレメント・リストで指定された１つまたは複数のエレメントがエレメント・リストに指定された順序でユーザー・プログラムとデータ・ベースとの間で受渡しされる。従って，ユーザーはさらにこのエレメントの順序や配置を整えたり，追加，削除といったことを行なう必要はない。

ユーザー・プログラムにデータが渡された後はプログラマーは，ホスト言語を用いて自分の好きなように，論理演算や数値演算等の処理を行なうことができる。ということはホスト言語はデータ処理の目的によってそれぞれに選ばれる言語であり，その処理の仕方は適用分野によってさまざまである。ＴＯＴＡＬ　ＤＢＣＬはホスト言語で書かれた枠組の中で，データ・ベースを相手に指定されたデータをやりとりする部分を受持つ。その意味でＴＯＴＡＬ　ＤＢＣＬは「データ・ベースの言語」であるということができる。

ＴＯＴＡＬは，そのＤＢＤＬ（データ・ベース定義言語）とＤＢＣＬ（データ・ベース命令言語）の機能を使って，完璧な総合データ・ベースを，どのアプリケーション・プログラマーにも，どんなホスト言語で書かれたプログラムからで

も，利用できるようにする。ＴＯＴＡＬは，データ・ベースに対する要求が変ればそれに応じた新しいエレメント，新しいデータ・セット，新しいデータ間の関係などを追加したりまたは削除したりしながらも，既存のプログラムには何の影響も与えることなく，要求されるエレメントをアプリケーションプログラムに渡すことができる。

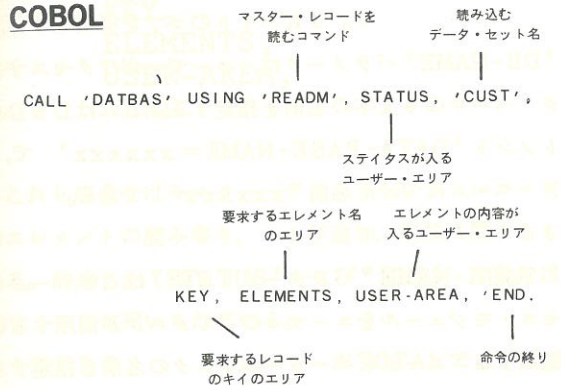


図６－２
データ・ベース・コマンド言語(DBCL)ステートメント例

ＴＯＴＡＬ　ＤＢＣＬはホスト・プログラミング言語のCALL機能を使ってＴＯＴＡＬを働かせる。図６－２参照。CALL・ステートメントが出されるとコントロールはＴＯＴＡＬに移り，ＴＯＴＡＬはパラメータ・リストを調べて実行すべき機能および対象となるデータを決める。

ＤＢＣＬ　ＣＡＬＬ命令の中のパラメータは，ユーザーのプログラムの中で定義されているコンスタントあるいはエリアの名前であり，それらの各々がＴＯＴＡＬの必要とするパラメータの実際の値や場所を指定するのである。どのサブ・プログラムがＣＡＬＬされるかによってパラメータ・リストはある定った順序で表示されなければならない。この章にでてくるコマンドの説明に記載しているパラメータ記入の順序は厳密に守られなければならない。

コントロールがＴＯＴＡＬからアプリケーション・プログラムに戻ると，ＴＯＴＡＬが実行した結果を示す情報が"ＳＴＡＴＵＳ"エリアにセットされる。実行にエラーがなければ，ステイタス・コードは"＊＊＊＊"がセットされる。もし実行にエラーがあれば，必要ならばデータ・セットはデータ操作実行前の状態に戻され，併せてエラーの原因を示す適当なステイタス・コードがセットされる。

ステイタス・コードの例として次のようなものがある。

＊＊＊＊	実行は正しく完了した。
ＦＮＴＦ	指定された名前のデータ・セットがない。従って命令も実行されていない。
ＩＰＡＲ	パラメータ・リストが正しくない。従って命令も実行されていない。

ＤＢＣＬ　ＣＡＬＬ命令の最後のパラメータは常に"END."である。

コマンド概説

ＴＯＴＡＬのＤＢＣＬを次の順序で概説する。

1. 実行開始の合図（ＳＩＧＮ－ＯＮ）
   (1) ＴＯＴＡＬ
   (2) ＭＰＴＯＴ
   (3) ＱＵＥＳＴ
2. データ・セットのオープン
   (1) ＯＰＥＮＭ
   (2) ＯＰＥＮＶ
3. マスター・データ・セットへのダイレクト・アクセス
   (1) ＲＥＡＤＭ
   (2) ＷＲＩＴＭ
   (3) ＡＤＤ－Ｍ
   (4) ＤＥＬ－Ｍ
4. バリアブル・データ・セットへのダイレクト・アクセス
   (1) ＲＥＡＤＶ
   (2) ＲＥＡＤＲ
   (3) ＲＥＡＤＤ
   (4) ＷＲＩＴＶ
   (5) ＤＥＬＶＤ
   (6) ＡＤＤＶＣ
   (7) ＡＤＤＶＡ

(9) ADDVR

5, シリアル・アクセス

(1) SEQRM

(2) SERLV

(3) SEQRV

(4) SEQWV

6. シリアル・リセット

(1) RESTM

(2) RESTV

7. データ・セットのクローズ

(1) CLOSM

(2) CLOSV

8. RRNの算出

(1) RQLOC

9. 実行終了の合図（SIGN-OFF）

(1) DEQUE

実行開始の合図（SIGN-ON）

```
CALL 'DATBAS' USING {'TOTAL'}
                    {'MPTOT'} ,
                    {'QUEST'}
     STATUS,
     DB-NAME,
     TASK-NAME,
     'END.'.
```

これらはTOTAL DBCLの実行開始を合図するコマンドである。これらのコマンドはDOS TOTAL 4では使わないが，OS TOTAL 4または5／6を使うときはこれらの内の1つを最初のコマンドとして出さなければいけない。

OS TOTAL 4または5／6を使うとき，第1番目のパラメータで，ユーザーが使用しようとするデータ・アクセス・モジュールを指定する。それらには次の3つがある。

TOTAL　このモジュールはすべてのデータ・マネジメント機能 … すなわち読み込み，書き込み，追加および削除 … をもつ。TOTAL 4では8K，TOTAL 5／6では14Kバイトのコアを必要とする。

MPTOT　このモジュールは読み込みおよび書き込み機能のみをもつ。追加および削除機能はないから，リンケージ・パスの更新はできない。5Kバイトのコアを必要とする。

QUEST　このモジュールは読み込み機能のみをもつ。3Kバイトのコアを必要とする。

"DB-NAME"パラメータは，ユーザーがアクセスするデータ・ベースの6文字の名前を指定する。これはDBDステートメント"DATA-BASE-NAME＝xxxxxx" で，データ・ベースにつけた名前"xxxxxx" である。

"TASK-NAME"パラメータ（8桁）はこのデータ・アクセス・モジュールをユーザーのどのタスクが使用するのかを識別するユニークなユーザーのタスクの名前を指定する。

データ・セットのオープン

```
CALL 'DATBAS' USING {'OPENM'} ,
                    {'OPENV'}
     STATUS,
     FILE,
     'END.'.
```

ユーザーはデータ・セットをアクセスする前に必ず当該データ・セットをオープンしておかなければならない。マスターおよびバリアブル・データ・セットをオープンするためのコマンドをここに示す。

"FILE"パラメータには，オープンするデータ・セットの名前（4バイト）を指定する。

マスター・データ・セットへのダイレクト・アクセス

```
CALL 'DATBAS' USING { 'READM' }
                    { 'WRITM' }
                    { 'ADD-M' }
                    { 'DEL-M' } ,
     STATUS,
     FILE,
     KEY,
     ELEMENTS,
     USER-AREA,
     'END.'.
```

これらの命令はマスター・データ・セット・レコードの中のエレメントの読み書き，および追加，削除に関するものである。前述した"STATUS"および"END."の他に以下のパラメータが用いられている。

FILE	ユーザーがTOTALでアクセスしようとしているマスター・データ・セットの名前。
KEY	オペレーションを行なおうとしている特定レコードを指定するキイの名前。TOTALはこのキイをもとにレコードのアドレスを計算する。
ELEMENTS	読み書きしようとするエレメントを指定する。READMとWRITM命令の場合は，読み書きするエレメントの名前を指定する。ADD-M命令の場合，指定されていないエレメントはすべてブランクにしてレコードを追加する。DEL-M命令の場合にもこのパラメータ指定は必要であるが，この場合にはTOTALは当該レコード全体を削除するので実際上の意味はない。
USER-AREA	ユーザー・プログラム内のエリア指定で，そこにTOTALの読んだエレメントを入れたり，ここからレコードに書き込むべきエレメントをとったりするエリアである。

これらのコマンドが出されたとき，TOTALは"KEY"で指定された値を"ランダム化"して，レコードのアドレスを計算し，それの読み込み，書き込み，追加または削除を行なう。図6－3参照。

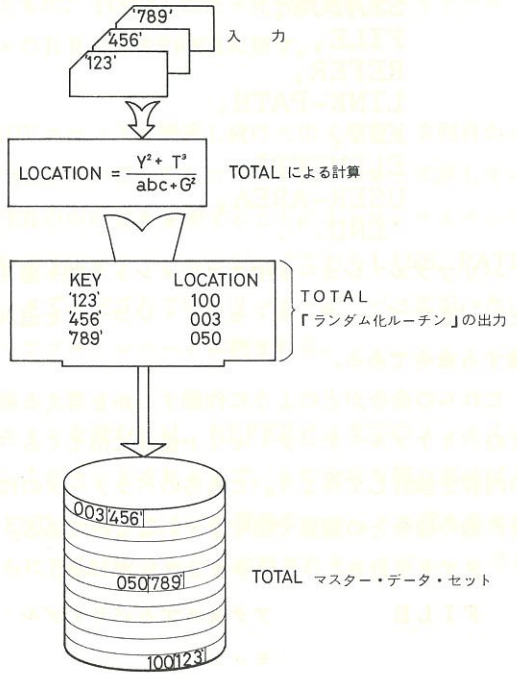


図6－3　ランダム化ルーチン

バリアブル・データ・セットへのダイレクト・アクセス

```
CALL 'DATBAS' USING { 'READV' }
                    { 'READR' }
                    { 'READD' }
                    { 'WRITV' }
                    { 'DELVD' }
                    { 'ADDVC' }
                    { 'ADDVB' }
                    { 'ADDVA' }
                    { 'ADDVR' } ,
     STATUS,
     FILE,
     REFER,
     LINK-PATH,
     KEY,
     ELEMENTS,
     USER-AREA,
     'END.'.
```

バリアブル・レコード中のエレメントを読み書きする命令と，バリアブル・データ・セット・レコードを追加または削除する命令である。

これらの命令がどのように作動するかを考える前に，すべてのバリアブル・データ・セット命令に出てくるパラメータの内容を検討してみよう。これらのパラメータのほとんどは既に他の命令との関係で出てきているものである。

STATUS	ステイタス・コード。
FILE	アクセスするバリアブル・データ・セット名。
ELEMENTS	読み書きするエレメント。 追加命令のときはTOTALはこのパラメータにないエレメントをすべてブランクにしてレコードを追加する。 削除命令のときは無視される。
USER-AREA	TOTALがエレメントをやり取りするためのユーザー・プログラム内のエリア。
END	命令の終了を示す。

残りの3つのパラメーターは，

REFER	リファレンス
LINK-PATH	リンケージ・パス
KEY	コントロール・キイ

である。

この3つのパラメータは，バリアブル・データ・セットを処理する命令のときは特別な意味を持つので，次にそれぞれを説明する。

既に他の章で述べたように，バリアブル・レコードは独立して存在しているものではない。むしろ各バリアブル・レコードは1つまたは複数のマスター・レコードに「従属」しているといえる。図6－4参照。バリアブル・レコードのチェインの最初と最後を見つけ出すのに，TOTALは次の情報を必要としている。

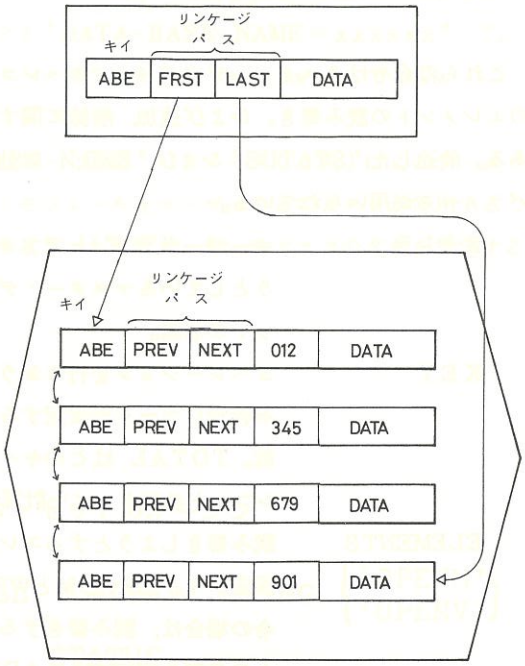


図6－4 リンケージ・パス

1. そのバリアブル・レコード・チェインをもっているマスター・レコードのキイ。
2. マスター・レコードをバリアブル・レコード・チェインに結びつけているリンケージ・パス。このマスター・レコード内のリンケージ・パスにはバリアブル・レコード・チェインの最初と最後のバリアブル・レコードのRRNが書かれている。TOTALはこれらのレコードのアドレスを算出するために，このRRNを使用する。

従って，バリアブル・レコードをアクセスするTOTAL DBCL 命令には，それらのバリアブル・レコードをもつマ

スター・レコードのキイと，そのマスター・レコードとバリアブル・レコードを関連づけるリンケージ・パス名とに関するパラメーターが含まれていなければならない。もちろん，この名前はこのデータ・ベースの DBMOD で定義されたものである。である。

" REFER " パラメータの役割は，TOTALに対しバリアブル・レコード・チエインの中で，TOTALにアクセスすべきレコードの所在を知らせることである。このパラメータについてはもう少し解説が進むと，よりはっきりしてくると思う。

(1) ＲＥＡＤＶ

READVコマンドはバリアブル・データ・セットからレコードを読み取り，そのレコードの中の指定されたエレメントを抽出し，その内容をユーザー・プログラムに渡す仕事を行なう。

このコマンドは指定されたリンケージ・パスに従って，バリアブル・レコードを論理的に正順に（上から下へ）読む。図６－５参照。全チエインを読むには，まずREFERパラメータにリンケージ・パス名の終りの４文字 " LKxx "を入れ，READVコマンドを出す。TOTALは，ＫＥＹパラメータの値を使って，マスター・レコードをアクセスし，このマスター・レコードのリンケージ・パス・エリアから最初のレコードのＲＲＮを受取る。このＲＲＮによって，TOTALはこのバリアブル・レコードを読み，その中からELEMENTS パラメータによって指定されたエレメント部の内容を，USER-AREAパラメータによって指定されたユーザー・プログラムの中のエリアへ渡す。コマンドの終了とともに，TOTALはたった今読まれたバリアブル・レコードのＲＲＮをREFERに戻す。

READVコマンドを繰返し続けて出す場合，２回目からはTOTALが前のＣＡＬＬコマンド終了のあとに戻しているREFERの中の値を使用することによって チエインの中の次のバリアブル・レコード，すなわちLINK-PATHパラメータで指定されているリンケージ・パスに沿った，次のバリアブル・レコードが読まれる。

全チエインを読むには，REFERに４文字のメッセイジ " END. " がセットされるまで，この命令を繰り返せばよい。" END. " はチエインの最後のレコードが読まれた後に，さらにREADVコマンドが出されたときにセットされる。

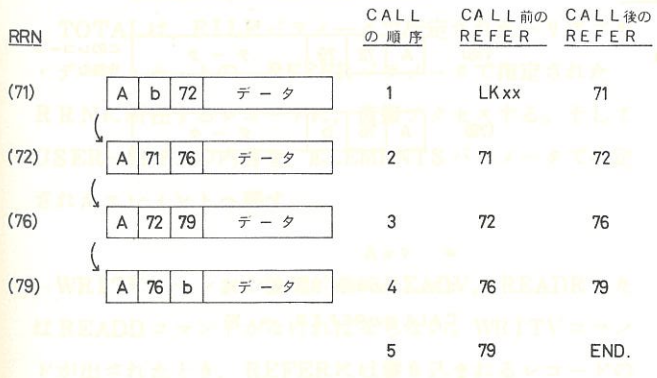


図６－５ READV (Read Variable Forward)

(2) ＲＥＡＤＲ

READRコマンドは，バリアブル・データ・セットからレコードを読み，そのレコードの中の必要とするエレメントを抽出し，それらをユーザー・プログラムに渡すのであるが，このコマンドはレコードの読み方が指定されたリンケージ・パスに従って，バリアブル・レコードを論理的に逆順にさかのぼって読む。図６－６参照。全チエインを読むには，まずユーザーは，REFERパラメータに"LKxx"を入れREADRコマンドを出す。TOTALはＫＥＹパラメータの中の値を使ってマスター・レコードをアクセスし，LINK-PATH によって指定されたチエインの，最初と最後のレコードのＲＲＮを受取る。TOTALはそのバリアブル・レコードを読むために，チエインの中の最後のレコードのＲＲＮを使用し，ELEMENTS パラメータによって指定されたエレメントをレコードから，USER-AREAパラメータによって指定されたユーザー・プログラムのエリアに移す。コマンドの終了とともに，TOTALはたった今読まれたバリアブル・レコードのＲＲＮをREFERに移す。

READRコマンドを繰り返し，前のCALLのときＴＯＴＡＬによって戻されたREFERの値を使うことによって，チエインの中の論理的に直前のバリアブル・レコードが読まれる。READRコマンドを続けて出すことによって…毎回，TOTALが前のCALLのあとに戻すREFERの中の値を使用することによって … 全チエインを最後尾から先頭まで読むことができる。

ユーザーがチエインの中の先頭のレコードを読んだあとさらにREADRコマンドを出したときは，TOTALはＲＥＦＥＲに４文字のメッセイジ" END. "を戻す。これは全チエインが読まれたことを示す。

(3) ＲＥＡＤＤ

READDコマンドは，ＦＩＬＥ パラメータによって指定されたバリアブル・データ・セットから，REFERパラメータに含まれているＲＲＮで示される場所にあるレコードを，直接読む。ELEMENTS パラメータに指定されたエレメントは，レコードより抽出され，ユーザー・プログラムのUSER-AREAに移される。

READD命令のときは，REFERは二進数表示のＲＲＮを指定しなければならない。図６－７参照。従ってＲＥＡＤＤコマンドを使用するときは，"LKxx" の形でＲＥ-

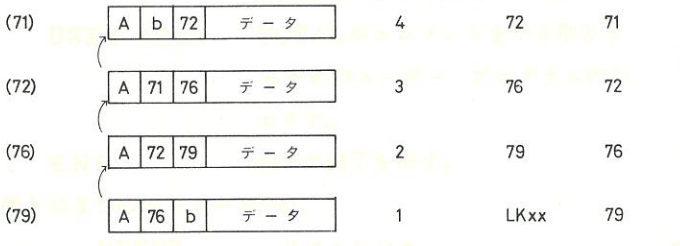


図６－６ READR (Read Variable Reverse)

図６－７
READD (Read Variable Direct)

FERの値を指定することは許されない。CALLが完了したとき，TOTALは同じRRNをREFERに戻す。すなわち，RRNの値は変らない。従って，アプリケーション・プログラムはこのRRNの値を起点にして，このリンケージ・パスにそって，次のどんな命令でも続けてゆくことができる。

READDコマンドを使用するときに，ユーザーはどのようにしてREFERに入れるべきRRNを知ることができるか？

それは次の2つのいずれかによることが多い。

* リンケージ・パスに従って処理中のプログラムが何かの都合で中断したあと，再び中断点から処理を再開しようとする場合には，RRNは既に保持されている。
* 他のプログラムによって，既にRRNが得られている場合，これをなんらかのインプット手段でこれから実行しようとするプログラムに引渡すことができる。

(4) WRITV

WRITVコマンドは，現存するバリアブル・データ・セット・レコードを更新するのに使われる。TOTALはユーザーが指定したエレメントの内容を，ユーザーが指定したバリアブル・データ・セット・レコードへ書き込む。

TOTALは，FILEパラメータで指定されたバリアブル・データ・セットの，REFERパラメータで指定されたRRNに所在するレコードに，直接アクセスする。そしてUSER-AREAの内容を，ELEMENTSパラメータで指定されたエレメントへ移す。

WRITVコマンドの論理的前にREADV，READRまたはREADDコマンドがなければならない。WRITVコマンドが出されたとき，REFERには書き込まれるレコードのRRNが入っていなければならない。このRRNの値は，WRITVコマンドに先行するREADV，READRまたは，READDコマンドの実行終了時にREFERにセットされる。REFERの値は"LKxx"の形であってはいけない。

CALLが終了するとき，TOTALは同じRRN値をREFERに戻す。

WRITVコマンドはリンケージ・パス，コントロール・キイまたはレコード・コードに書き込む（すなわちこれらのフィールドを更新する）場合に使用することはできない。

図6－8はREADVとWRITVとが一緒に使われる場合の例である。ここでは第2および4番目のレコードが，WRITVにより更新される。

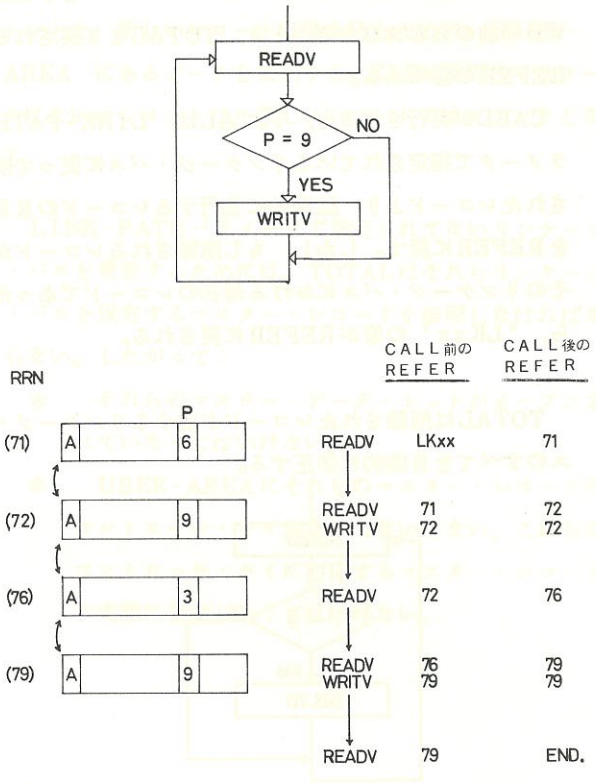


図6－8 WRITV (Write Variable)

(5) DELVD

DELVDコマンドは，FILEで指定されたバリアブル・データ・セットから，REFERで指定されているRRNにあるバリアブル・レコードを，論理的かつ物理的に削除する。削除は，そこにブランクを入れることにより，当該レコードは消え，空いたスペースは再使用のために直ちに利用できる。

CALLが行なわれるときは，REFERに削除されるべきレコードのRRNがなくてはいけない …… つまりDELVDの前のREAD命令のとき，TOTALより戻されるREFERの値である。

CALLが実行されると，TOTALは，LINK-PATHパラメータで指定されているリンケージ・パスに従って削除されたレコードより，論理的に先行するレコードのRRNをREFERに戻す。しかし，もし削除されるレコードが，そのリンケージ・パスにおける最初のレコードであったなら，"LKxx" の値がREFERに戻される。

TOTALは削除されたレコードが属するリンケージ・パスのすべてを自動的に修正する。

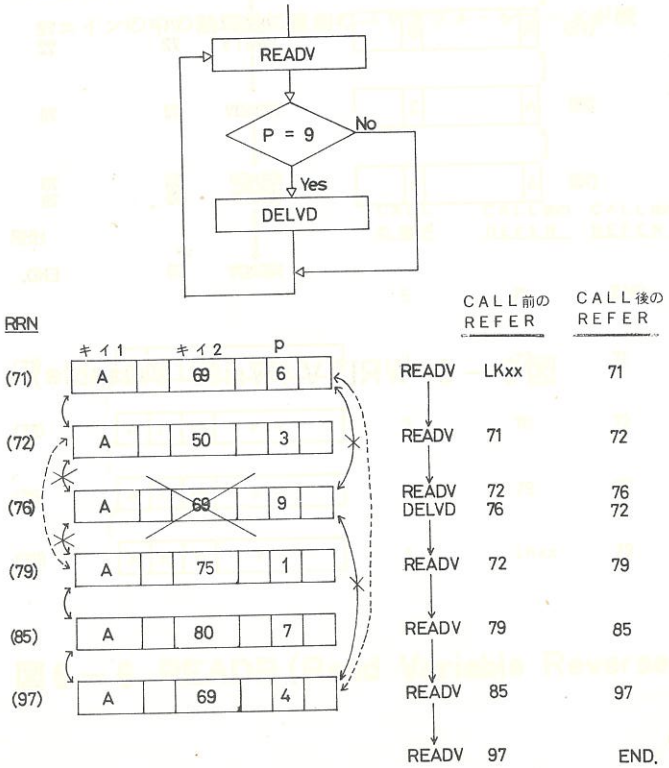


図6－9 DELVD (Delete Variable)

図6－9は，READVとDELVDとが一緒に使われる場合の例である。ここでは第3番目のレコードがDELVDにより削除され，それに伴いリンケージ・パスが自動的に修正される。点線は修正をうけたリンケージ・パスの修正後を示す。

(6) ADDVC

ADDVCコマンドは，FILEパラメータで指定されたバリアブル・データ・セットに，1つのレコードを追加する命令である。新しいレコードは，DBMODでこのレコードのために定義されているすべてのリンケージ・パスの最後に追加される。図6－10参照。

TOTALにこのCALLを出すとき，REFERにはリンケージ・パスの最後の4文字"LKxx"がなくてはいけない。

CALLが実行されるとREFERには新しく追加されたレコードのRRNが入る。

TOTALは，USER-AREAにある内容をELEMENTSパラメータで指定されたとおり，エレメントとして，新

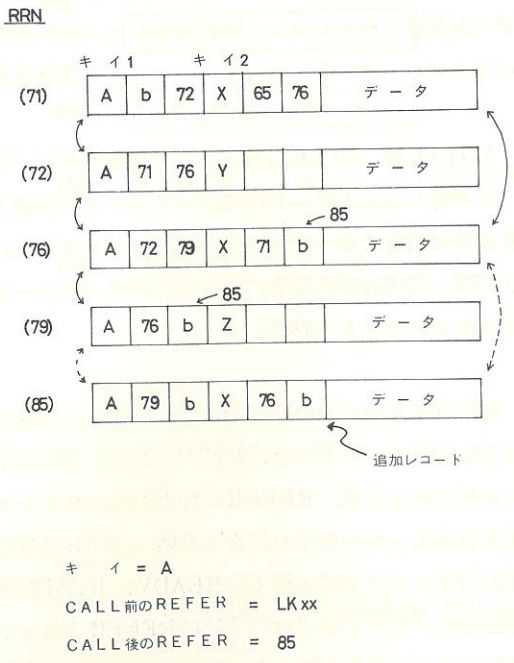


図6－10 ADDVC (Add Variabl Continue)

しいレコードを書き込む。ELEMENTS パラメータで指定されていないエレメントは，新しいレコードではブランクとなる。

LINK-PATHパラメータで指定されていないリンケージ・パスを更新するためには，TOTALはそれらリンケージ・パスを保有するマスター・レコードを参照しなければならない。したがって；

＊　それらのマスター・データ・セットがオープンされていなくてはいけない。

＊　USER-AREAにこれらのマスター・レコードのコントロール・キイがなくてはいけない。これらのコントロール・キイに対応するマスターレコードが実際に存在していなくてはいけない。

(7)　ADDVA

ADDVAコマンドは，FILEで指定されたバリアブル・データ・セットに，1つのレコードを追加する。REFERに指定されているRRNの指すレコードの，論理的後の，LINK-PATHで指定されているリンケージ パスに，レコードが追加される。図6－11参照。もしREFERに，"LKxx"の形で値が入っていると，新しいレコードはそのリンケージ・パスの最後に追加される。

CALLが実行されるとREFERには新しいレコードのRRNが入る。

新しいレコードは，そのレコードに定義されている他のすべてのリンケージ・パス（図では2番目のリンケージ・パス）の最後に追加される。

TOTALは，ELEMENTSパラメータで指定されたエレメントを，新しいレコードに書き込むために，USER-AREA にあるデータを使用する。ELEMENTSパラメータにないエレメントは，新しいレコードではブランクとなる。

LINK-PATHパラメータで指定されてないリンケージ・パスを更新するためには，TOTALはそれらリンケージ・パスを保有するマスター・レコードを参照しなければならない。したがって；

＊　それらのマスター・データ・セットがオープンされていなくてはいけない。

＊　USER-AREAにそれらのマスター・レコードのコントロール・キイがなくてはいけない。これらのコントロール・キイに対応するマスター・レコードが実際に存在しなくてはいけない。

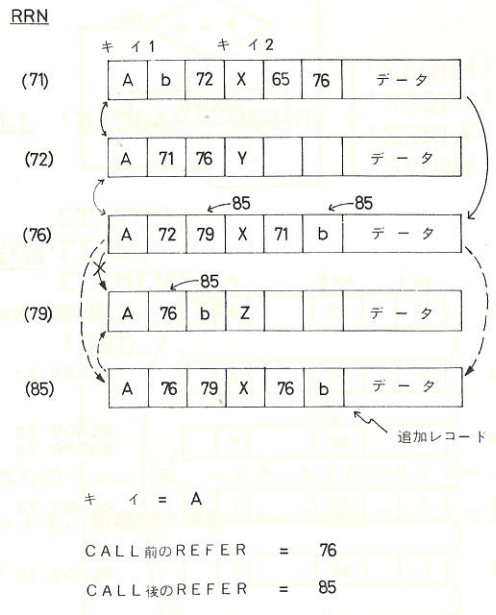


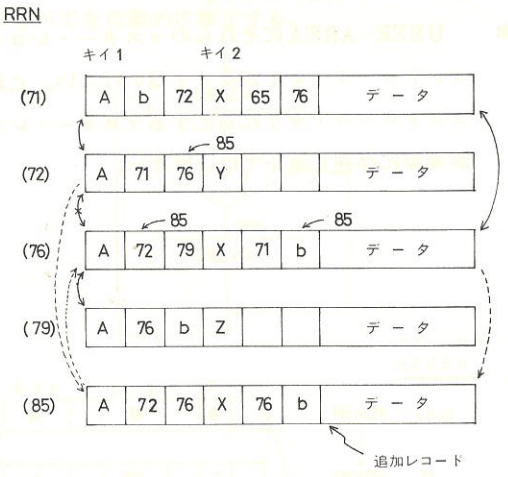
図6－11 ADDVA（Add Variable After）

(8) ＡＤＤＶＢ

ADDVBコマンドは，ＦＩＬＥで指定されたバリアブル・データ・セットに，１つのレコードを追加する。REFERに指定されているＲＲＮの指すレコードの，論理的前の，LINK-PATHで指定されているリンケージ・パスに，レコードが追加される。図６－１２参照。もしREFERに，＂LKxx＂の形で値が入っていると新しいレコードは，そのリンケージ・パスの先頭に追加される。ＣＡＬＬが完了するとREFERには，新しいレコードの，ＲＲＮが入る。

新しいレコードは，そのレコードに定義されている他のすべてのリンケージ・パス（図では２番目のリンケージ・パス）の最後に追加される。

TOTALは，ELEMENTSパラメータで指定されたエレメントを，新しいレコードに書き込むために，USER-AREAにあるデータを使用する。ELEMENTSパラメータにないエレメントは，新しいレコードではブランクとなる。



キ　イ = A

ＣＡＬＬ前のＲＥＦＥＲ = 76

ＣＡＬＬ後のＲＥＦＥＲ = 85

図６－12

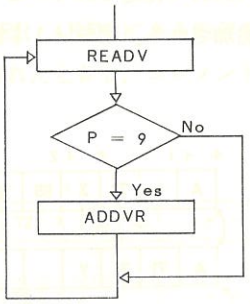
ADDVB (Add Variable Before)

(9) ＡＤＤＶＲ

ADDVRコマンドは，現存するバリアブル・データ・セット・レコードのキイとエレメントを更新する。キイが更新される場合は，それに伴い関連するリンケージ・パスの付け変えをする。図６－１３参照。

TOTALは，第１に，ＦＩＬＥで指定されたバリアブル・データ・セットの，REFERで指定されているＲＲＮにあるバリアブル・レコードのすべてのリンケージ・パスを切断する。第２に，ＵＳＥＲ-ＡＲＥＡにある内容を，ＥＬＥＭＥＮＴＳで指定されたとおりエレメントとして，そのレコードに書き込む。USER-AREAには，キイを含めることができる。第３に，そのレコードのキイ（キイの更新があれば，更新後のキイ）に従って，そのレコードに関するすべてのリンケージ・パスを論理的に正しく再結合する。

レコードの物理的位置は変わらない。そのレコードは，それに定義されているベース・リンケージ・パス以外の他のすべてのリンケージ・パス（図では２番目のリンケージ・パス）の最後に追加される。



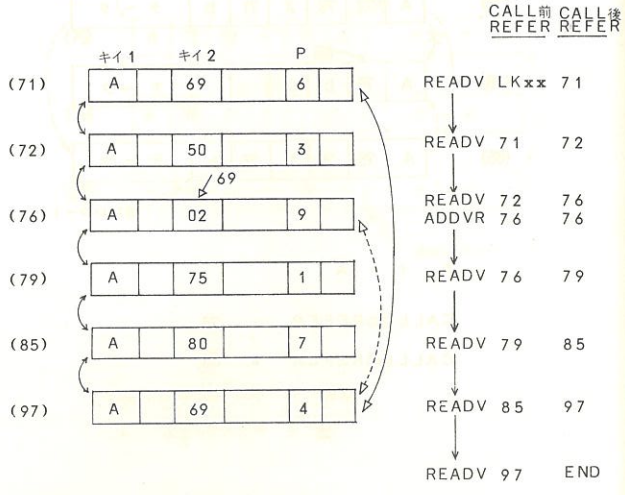


図６－13 ADDVR(Add Variable Replace)

ELEMENTSで指定されていないエレメントは更新を受けない，従ってこのコマンドの実行前と同じである。

CALLが行なわれるときは，REFERに更新を受けるレコードのＲＲＮがなくてはいけない……つまり，ADDVRの前のREADコマンドのとき，ＴＯＴＡＬより戻されるREFERの値である。CALL が実行されても REFERの値は変わらない。

ADDVRコマンドで，ベース・リンケージ・パスのキイを更新できない。それを更新するにはDELVDとADDVコマンドを用いる。

注：「ベース・リンケージ・パス」とは，レコードの中の最初の……すなわち一番左の……リンケージ・パスである。

シリアル・アクセス

```
CALL 'DATBAS' USING { 'SEQRM' }
                    { 'SEQRV' }
                    { 'SERLV' }
                    { 'SEQWV' } ,
     STATUS,
     FILE,
     ELEMENTS,
     USER-AREA,
     'END.'.
```

これらのコマンドは，マスターおよびバリアブル・データ・セットを，物理的な順番（キイの順番でない）でアクセスする。

ＳＥＱＲＭ　マスター・レコードを物理的順番に読む。

ＳＥＲＬＶ　バリアブル・レコードを物理的順番に読む。

ＳＥＱＲＶ　バリアブル・レコードを論理的チエインの順番に読む。

ＳＥＱＷＶ　直前のSEQRVまたはSERLVコマンドにより読まれたバリアブル・レコード内に，エレメントを書き込む。マスター・レコードに対しては，WRITMコマンドが同様の機能をもっている。

SEQRMコマンドはマスター・データ・セットをシリアルに読む …… すなわち物理的順序であり，論理的またはキイの順序ではない。

SERLVコマンドはバリアブル・データ・セットに対して同様の機能を有する。図６－１４参照。最初のSEQRMまたはSERLVの実行時に，TOTALはインターナル・カウンターに，データ・セットの中の最初のレコードのＲＲＮをセットする。

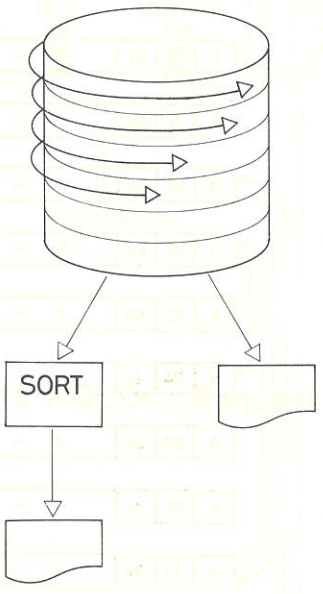


図６－14　SEQRM (Serial Read Master)
SERLV (Serial Read Variabe Without Linkage)

次のSEQRMまたはSERLVコマンドで，TOTALはこのカウンターに1を加え，それ（ＲＲＮ＋１）に対応するレコードを検索する。

TOTALはデータを含んでいるレコードのみをユーザーのプログラムに渡す。ブランク・レコードとTOTALのコントロール・レコードはバイパスされる。TOTALがレコードを検索すると，ELEMENTSパラメータによって指定されたエレメントの内容が，ユーザー・プログラムに渡される。

TOTALがデータ・セットの終りに達したとき，STATUSに" END. "がセットされる。

SEQRVコマンドは，バリアブル・データ・セットをシリアルに読む …… すなわち物理的順序で，キイの順序ではない。図６－１５参照。TOTALはそれぞれのチエインの最初から最後まで「ベース・リンケージ・パス」に従ってレコードを検索する。

注：「ベース・リンケージ・パス」とは，レコードの中の最初の … すなわち一番左の … リンケージ・パスである。

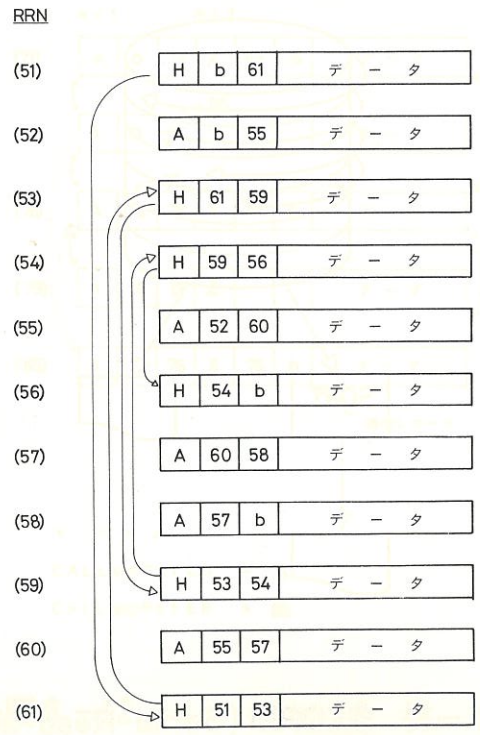


図６－１５
SEQRV (Serial Read Variable with Linkage)

最初のSEQRVコマンドで，TOTALは，インターナル・カウンターをデータ・セットの中の最初のレコードのＲＲＮにあわせる。TOTALは，次にこのレコードを検索し調べる。もし，このレコードがデータを含んでおり，そのベース・リンケージ・パスの最初のレコードであるならば，TOTALはELEMENTS パラメータによって指定されたエレメントをレコードより抽出し，それらを USER-AREA へ移す。

次のSEQRVコマンドにより，TOTALはこのチエインの処理をつづける。チエインの終りに達したとき，TOTALはチエインの最初のレコードに戻り，２番目のチエインの最初をシリアルに探がす。

データを含んでいるレコードのみが，ユーザー・プログラムに渡される。ブランク・レコードとTOTALのバリアブル・コントロール・レコードはバイパスされる。

TOTALがデータ・セットの終りに達したとき，STATUSに" END. "がセットされる。

シリアル・リセット

```
CALL 'DATBAS' USING {'RESTM'} ,
                    {'RESTV'}
     STATUS,
     'END.'.
```

RESTMおよびRESTVコマンドはインターナル・カウンター …… TOTALがSEQRM，SERLVおよびSEQRVコマンドでデータ・セットをシリアルに処理するときに使う…… をゼロにする。したがって次のSEQRM，SERLVまたはSEQRVコマンドは，マスターまたはバリアブル・データ・セットの最初のレコードを検索する。

データ・セットのクローズ

```
CALL 'DATBAS' USING {'CLOSM'} ,
                    {'CLOSV'}
     STATUS,
     FILE,
     'END.'.
```

これらのコマンドは，TOTALおよびオペレーティング・システムに対して論理的にマスター・データ・セットのクローズをするための命令を出す。

オープンしたデータ・セットは，すべてクローズしなくてはいけないことをプログラマーは覚えておくことが大切である。データ・セットに対するアクセスがすべて完了したら，できるだけ早く，それぞれのデータ・セットをクローズすべきである。

ＲＲＮの算出

```
CALL 'DATBAS' USING 'RQLOC',
     STATUS,
     FILE,
     KEY,
     USER-AREA,
     'END.'.
```

このコマンドは，ＦＩＬＥパラメータにより指定されたマスター・データ・セットの，ＫＥＹパラメータによって指定されたレコードのロケーション（すなわちＲＲＮ）を算出する。図６－１６参照。TOTALはＫＥＹの内容で「ランダム化」し，計算されたＲＲＮをUSER-AREAに入れる。Ｉ／Ｏオペレーションは実行されない。データ・セットはオープンされる必要はない。

アプリケーション・プログラムがマスター・データ・セットを大量に …たとえば３０％… アクセスする場合には，トランザクションをできるだけ … キイの値によって決まるＲＲＮの順番に … ソートしてから入力すれば大きなメリットが得られるであろう。マスター・レコードはランダムな順序に格納されているのであるから，意味のある唯一の処理順序は与えられたキイによって決まる相対物理的ロケーション（ＲＲＮ）の順序である。

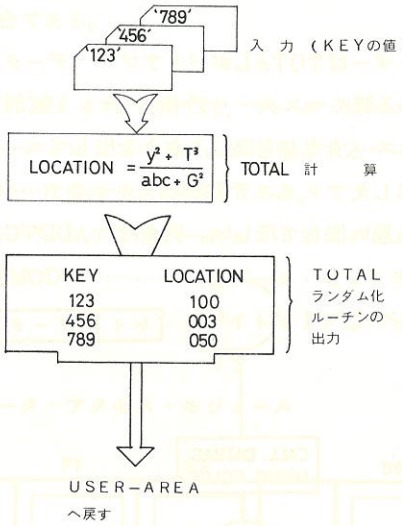


図６－16 RQLOC (Request Location)

RQLOCコマンドにより，プログラマーはレコードのキイを知っていれば，どんなマスター・レコードのロケーションでもみつけることができる。ユーザーはインプット・トランザクション・ストリームの中のそれぞれのマスター・レコードのＲＲＮをみつけるために，RQLOCを使用し，これらのＲＲＮでトランザクション・ストリームをソートすることができる。これはＤＡＳＤアームの動きを最少にし，バッファの利用を最大にすることによって，シークエンシャル・バッチのアプリケーションの処理時間をいちぢるしく削減する。図６－１７参照。

処理されるトランザクションの量の多少によって，RQLOC処理および前もってソートを行なうことに要する時間がＤＡＳＤアーム・コンテンション時間より多いか少ないか，それぞれの状況に応じて判断されなければならない。

ユーザーはTOTALがバリアブル・データ・セットを処理している間にマスター・データ・セットに対して「かくれた」アクセス…すなわちCALL命令を出してユーザーがはっきりと要求したアクセスではないアクセス …… を行なっていることを思い出してほしい。例えば，ADDVC，READV，

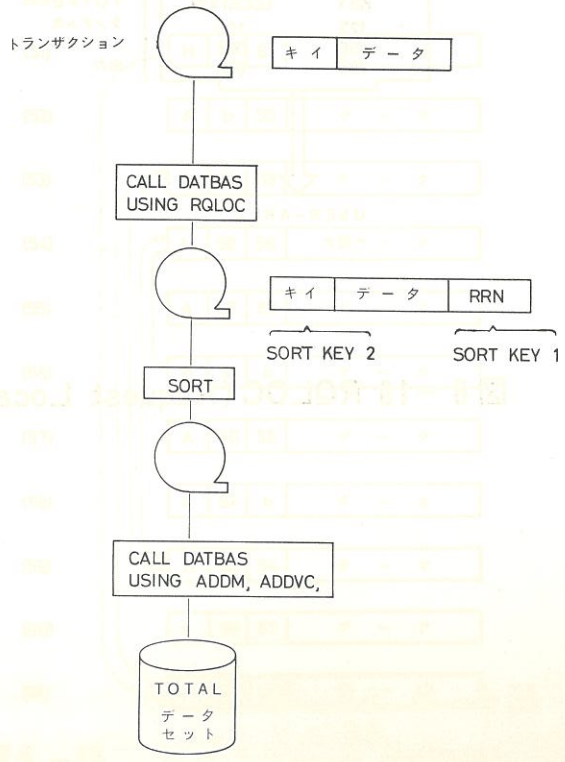


図６－17 RQLOCの使用例

READRコマンドの場合，TOTALはマスター・データ・セットにアクセスしてバリアブル・データ・セットのＲＲＮを取り出している。これらの「かくれた」アクセスがあるので，バリアブル・データ・セットをバッチ処理するときでさえ，RQLOCを使用した方がよい。

マスター・データ・セットがかなり多くの数のシノニムを含んでいる場合には，トランザクションをRQLOCの逆順にソートした方が有利であろう。

実行終了の合図（ＳＩＧＮ－ＯＦＦ）

```
CALL 'DATBAS' USING 'DEQUE',
     STATUS,
     TASK-NAME,
     'END.'.
```

これはTOTAL DBCLの実行終了を合図するコマンドである。OS TOTAL ４または５／６で用いられる。

OS TOTAL ４のもとでは，ユーザーのプログラムが，「サイン・オン」されているデータ・アクセス・モジュール …… TOTAL，MPTOTまたはQUEST …… に対して，このコマンドは，まだＩ／Ｏバッファに残っているデータをディスクに書き込み，DATBASをデータ・アクセス・モジュールから切離し，さらにユーザー・プログラムからDBMODを切り離させる。

OS TOTAL ５／６ のもとでは，TOTALがこの'DEQUE' コマンドを出したユーザー・プログラム（つまり，タスク）のためにホールドしているすべてのレコードを，このDEQUEコマンドでリリースし，TOTALとユーザー・プログラム間のコミュニケーションを断絶する。DEQUEによって，Ｉ／Ｏバッファにまだ残っているデータをディスクに書き込むことは行なわれない。TOTAL ５／６は，指定されたチェックポイント（QUIETインターバルと呼ばれる）のときでしかも，バッファに新しいデータをもってくる必要のあるときにのみ，Ｉ／Ｏバッファのデータをディスクに書き出す。

第７章　TOTALのオペレーション・モード

TOTALのオペレーション・モード概説

この章は，TOTALのそれぞれのバージョンがどのように実行されるかを説明する。

TOTALには，異ったオペレーション・モードで実行される，いくつかのバージョンがある。

バージョン	基本的オペレーション・モード
DOS TOTAL 4	IBM DOS オペレーティング・システムの下での，バッチのシングル・タスク処理
OS TOTAL 4	IBM OS オペレーティング・システムの下での，バッチのシングル・タスク処理
OS TOTAL 5／6	IBM OS オペレーティング・システムの下での，オンラインまたはバッチのマルチ・タスク，シングル・スレッド処理
OS TOTAL 5／6 S	IBM OS オペレーティング・システムの下での，オンラインまたはバッチのマルチ・タスク，マルチ・スレッド処理

TOTALの各バージョンは，それぞれ各モードのどれか１つで動くことを目的としてはいるが，これはそのモードでしか動かないということを意味しない。例えば，DOS またはOS TOTAL 4 は，バッチでもオンライン処理環境の下でも，マルチ・タスク処理にも使うことができる。

DOS TOTAL 4

図７－１は，いかにDOS TOTAL 4 が実行されるかを示している。TOTAL モジュールとDBMODは両方ともユーザーのアプリケーション・プログラムにリンク・エディットされる。

プログラムが何回実行されようとも，たった一回リンク・エディットすればよい。

リンク・エディットを再び実行しなければならないことがあるとすれば，その理由は，

＊ プログラムの再コンパイルを必要とするようなユーザー・アプリケーション・プログラム・ロジックの変更

＊ DBMODの再アセンブリーを必要とするようなデータ・ベースの変更

があった場合である。

DOS TOTAL 4 を使用して，データ・ベースを処理するアプリケーション・プログラムが必要とするパーテイションのサイズは　次のサイズの合計である。

＊ ユーザー・アプリケーション・プログラム

＊ DBMOD …… １データ・セット当り２００バイト＋Ｉ／Ｏバッファ・エリア

＊ データ・アクセス・モジュール

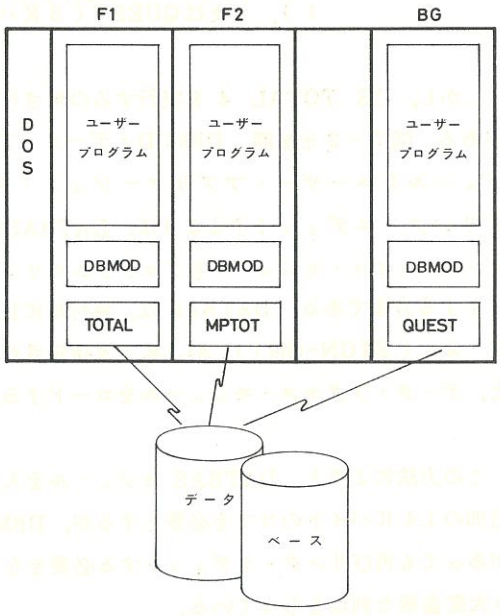


図７－１ DOS TOTAL 4

データ・アクセス・モジュールのサイズは，次の3つのモジュールのうちどのモジュールが使われるかによって異なる。

TOTAL　8Kバイトのコアを必要とする（レコードを追加・削除し，エレメントを読み書きする機能をもつ）。

MPTOT　約5Kバイトのコアを必要とする（エレメントを読み・書きする機能をもつ。レコードを追加・削除する機能をもたない）。

QUEST　約3Kバイトのコアを必要とする（エレメントを読むだけの機能をもつ）。

OS TOTAL 4

OS TOTAL 4は，DOS TOTAL 4と同様に使用される。DBMODとデータ・アクセス・モジュール… TOTAL，MPTOTまたはQUEST … の1つが，ユーザーのアプリケーション・プログラムとリンク・エディットされる。パーティションまたはリージョン・サイズは，次のサイズの合計である。

＊　ユーザーのアプリケーション・プログラム

＊　DBMOD …… 1データ・セット当り200バイト＋I／Oバッファ・エリア

＊　データ・ベース・アクセス・モジュール … TOTAL（8Kバイト），MPTOT（5Kバイト），またはQUEST（3Kバイト）

しかし，OS TOTAL 4を実行するのにさらに良い方法がある。図7－2を参照。DBMODとデータ・アクセス・モジュールをユーザー・アプリケーション・プログラムにリンク・エディットするよりは，DATBASとよばれるインタフェイス・モジュールをプログラムにリンク・エディットする方法である。DATBASは，実行時にDBCL「サイン・オン（SIGN-ON）」CALLにより指定されたDBMODと，データ・アクセス・モジュールをロードする。

この方法によると，DATBASモジュールを入れるために追加の1.4Kバイトのコアを必要とするが，DBMODに変更があっても再びリンク・エディットする必要をなくするという大変重要な利点をもっている。

OS TOTAL 5／6

TOTAL 5／6は，MFT，MVT，VS1，VS2などあらゆるIBM OSオペレーティング・システムの下での，バッチ，マルチ・プログラム処理およびオン・ライン・マルチ・タスク処理のために開発された。図7－3参照。オン・ライン・アプリケーションの場合，TOTAL 5／6は，CICS，TSO，IMS／DC，ENVIRON／1，INTERCOMM，TASK／MASTERなど，今日一般に使用されているすべてのテレプロセッシング・パッケージとインタフェイスを持っている。TOTAL 5／6はまた，BTAMやTCAMを用いて，ユーザーが開発した数多くのテレプロセッシング・システムの下で稼動する。

TOTAL 5／6の場合，DBMODおよびデータ・アクセス・モジュールは，個々のアプリケーション・プログラムとはリンクせず，またアプリケーション・プログラムが占めるベーティションやリージョンにもロードされない。その代り，TOTAL 5／6は，データ・ベースをアクセスするすべてのタスクまたはパーティションに対して独立したプロセッサーとして機能する。

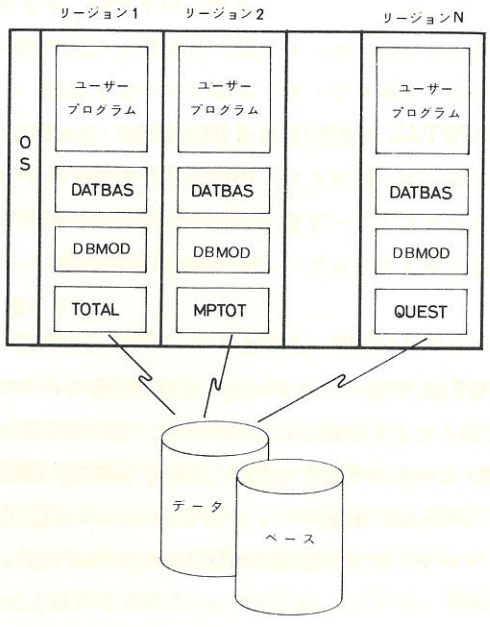


図7－2 OS TOTAL 4

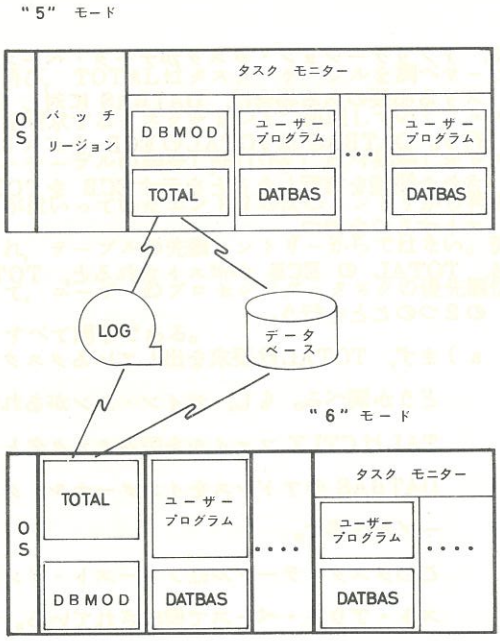


図7－3 OS TOTAL 5/6

TOTAL 5／6は次の2つのオペレーション・モードのいずれかの形で用いられる。

"5"モード…… TOTALはATTACH命令で，テレプロセシング域へ1つのサブ・タスクとして加えられ，同一域内の他のすべてのサブ・タスクにサービスを提供する。

"6"モード…… TOTALは，コンピュータ内のあらゆるタスクまたはサブ・タスクに，サービスを提供する特定リージョン（またはパーティション）内の1プログラムとして，EXEC ステートメントで稼動する。

TOTAL 5／6は，データ・ベースを集中的に制御し，どのユーザーにもその機能を自由に提供する。同時に共通データ・ベースをアクセスし，更新するタスクおよびサブ・タスクの数に制限はない。

データ・ベースの保全は，TOTAL 5／6がデータ・ベースのいかなる変化も集中的に時系列順にログへ記録することにより保証される。

"5"または"6"のいずれのオペレーション・モードも，TOTALタスク・リージョンまたは，パーティションとして以下のコア・スペースが必要である。

TOTAL　　14Kバイト

DBMOD ュール　1データ・セット当り200バイト＋I／Oバッファ・エリア

TOTALデータ・ベースをアクセスするユーザー・アプリケーション・プログラムそれぞれに，1400バイトのインタフェイス・モジュール（DATBAS）がリンク・エディットされる。

"6"モードの場合TOTALは，ECB（EVENT CONTROL BLOCKS）へのポスト（POST）と，パーティション間コミュニケーションを行なうためTYPE 1 SVC ルーチンを必要とする。

"6"モードの場合は，どんなタスクにABENDが生じてもデータ・ベースの物理的関係を損なわない。"5" モードの場合には，ユーザーのタスク・モニターがサブ・タスキングをどのように処理するかによって，ABENDの影響が異なってくる。

TOTAL 5／6とTOTAL 4，MPTOT，QUEST …… コピーが作られた場合はコピーの1つ1つ …… は，それぞれに自分のDATBAS モジュールをもち，それらは同時にそれぞれ実行することができる。

パーティション間のコミュニケーション

オペレーティング・システムおよびタスク・モニターからみれば，TOTALとTOTALデータ・ベースをアクセスする個々のユーザー・アプリケーション・プログラムは，それぞれ完全に独立したタスクである。TOTALおよびアプリケーション・プログラムはいずれも，普通のオペレーテング・システムのタスクまたはタスク・モニター（例えばCICS）のサブ・タスクとして稼動する。これは大へん重要な意味をもつことで，TOTALがＯＳのどんなバージョンの下でも稼動し，考えられるすべてのタスク・モニターとインタフェイスを持つ理由はここにある。

一方，TOTALはアプリケーション・プログラムに対し，何らかのコミュニケーション手段を持たなければならない。このコミュニケーションを確保するため CTLX ファイルがある。図７－４参照。CTLXは，４０バイトの非ブロック化レコードを収めた１シリンダーのTOTALマスター・データ・セットである。TOTALとアプリケーション・プログラムとのコミュニケーションは以下のように行なわれる。

1. TOTALが最初にスタートしたときTOTALは，
   a）DBMODのロード，ログ・テープのオープンなど種々のイニシアライゼーション機能を実行する。
   b）CTLX ファイルに自分の位置（ position ）を記録する。
   c）ECB（Event Control Block）をWAIT状態にして，"6" モードはオペレーティング・システムに，"5" モードではタスク・モニターに準備完了を知らせる。
2. アプリケーション・タスクから，サイン・オン・コマンドが出されると，DATBAS は，
   a）CTLX ファイルからTOTAL の位置を知る。
   b）アプリケーション・タスクの位置をCTLX ファイルへ記録する。
   c）TOTALのECB をポストし，TOTALにリクエストのあることを知らせる。
   d）TOTALがそのECB をポストするのを待つ。
3. アプリケーション・タスクがデータ・ベースをアクセスする必要のある場合は，DATBAS に対し CALL を行う。DATBAS はTOTALのECB をポストし，要求された機能を実行したことを示すECB をTOTALがポストするのを待つ。
4. TOTAL の ECB がポストされると，TOTALは次の２つのことを行う。
   a）まず，TOTALは要求を出しているタスクがあるかどうか調べる。もし，サイン・オンがあれば，TOTALはCTLX ファイルを読みタスク名とタスクのDATBAS のアドレスをインターナル・タスク・テーブルへ移す。
   このタスク・テーブルはファースト・イン，ファースト・アウト・ベースで構成されている。すなわち，最初に要求を出したタスクが最初にテーブルへ入れられる。タスク・テーブルには２１個のタスクまで入る。タスク・テーブルが一杯になった後に要求を出したタスクは，WAIT状態になり，他のタスクが終了するが，ABEND になるかしてテーブルのスペースの空くのを待つ。
   タスク・テーブルはTOTALを再コンパイルすることによりそのサイズを広げることができる。その場合１タスク分増えるごとに４８バイト必要となる。
   b）次に，TOTALはタスク・テーブルを調べサービスを要求しているタスクを見つけ出す。そのタスクの要求する機能を実行し，終了するとそのタスクの，ＥＣＢをポストする。

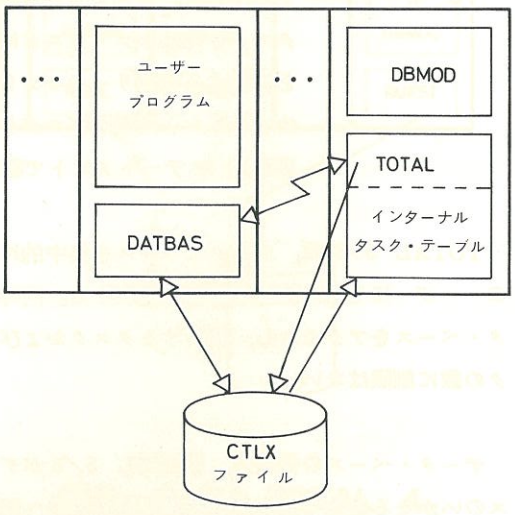


図７－４ CTLX ファイル

再び，TOTALはタスク・テーブルを調べサービスを要求するタスクを１つ１つ処理していく。タスク・テーブルの調査（polling）は今処理したタスクがはいっていたエントリの次のエントリから再開され，テーブルの先頭エントリーからではない。従って，ユーザーのプロセシング・タスクの優先順位はすべて同じである。

もし，タスク・テーブルに１つもサービスを要求するタスクがなければ，TOTALはWAIT ECBを送り，オペレーティング・システムまたはタスク・モニターに，他に何もすることがないことを知らせる。もちろんマルチ・プログラミングのもとではTOTALが何もすることがないということはほとんど起り得ないことではあるが。

第8章　データ・ベースの保護

データ・ベースは，高価な財産と同じように，保護されなければならない。この章では，下記に関するTOTALの機能について解説する。

＊　データ・ベースの中の，機密情報へのアクセスの制限

＊　予期できないデータ・ベース破壊の防止

＊　データ・ベースが壊れたときの復旧

データの機密…… データへのアクセスの制限

従来の順次処理システムでは，ファイルは通常，アプリケーションごとに作られる。従って，アプリケーションはそれぞれ自分個有のファイルを持ち，他のアプリケーションと，ファイルを共用することは稀である。そこで"データの機密"はたいして問題とされなかった。

"データ・ベース"の概念を取り入れてくると，データやファイルは中央に集められ，さまざまなアプリケーションが同じファイルにアクセスすることになる。従ってデータ・ベース・システムではしばしば機密情報へのアクセスを制限する必要が生ずる。

TOTALはこのような"データの機密保護"をエレメントのレベルで行う。

既に説明したように，TOTALがユーザー・アプリケーション・プログラムへ渡すのは，要求されたエレメントであって，当該レコードの全体ではない。図8－1参照。

ユーザーは，それ故，次の2つを満たさないかぎりエレメントをアクセスできない。

1　そのユーザーが使っているDBMODの中に，そのエレメントが定義されていること

2　そのユーザーはそのエレメントの名前 …
"mmmmxxxx"，または"vvvvxxxx"… を知っていること

従って，データ・ベースの設計者は機密エレメントへのアクセスを次のように制限することができる。

同じデータ・ベースに対して，複数のDBMODを定義するか，またはこれらの機密エレメントの名前を，このデータへのアクセスを許された人にのみ知らせる。

例えば，図8－2の"給与額"情報については，この情報へのアクセスを制限するために，データ・ベース設計者は次のいずれかを実施することができる。

1. 2つのDBMODを定義する。一般用の方は，最初から"N"エレメント名までのみを定義する。人事部専用のもう1つのDBMODには，エレメント"SALARY"も定義する。
2. または，データ・ベース設計者はただ1つのDBMODを定義するのであるが，エレメント名の記述書を2通りに作る。一般用の方は"SALARY"エレメントを省いておく。人事部の人にのみ"SALARY"エレメントを教えるのである。

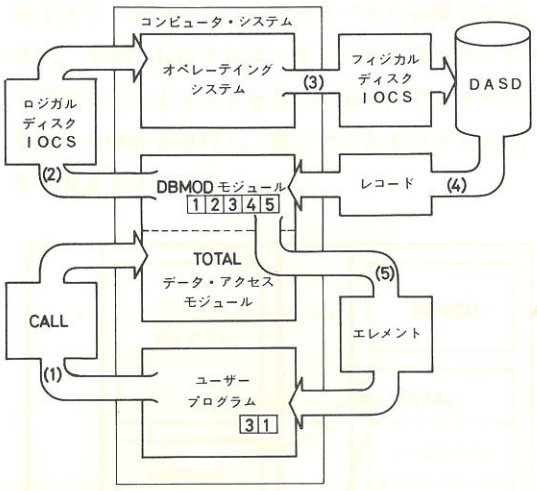


図8－1

エレメントを要求してから受け取るまで

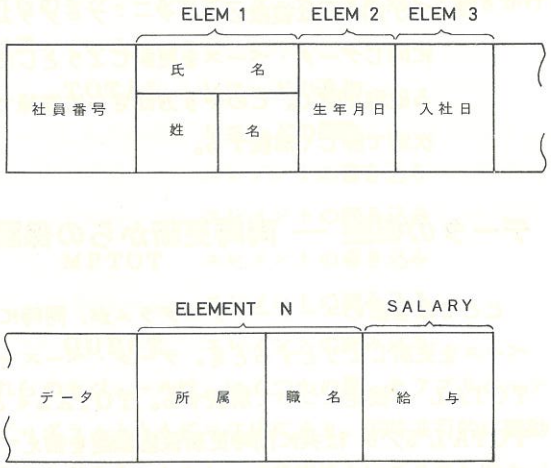


図8－2

SALARY エレメントの機密保護

この TOTALのやり方は，レコードやセグメント全体を機密保護する方式よりも，より大幅に効率的である。なぜなら本当に保護が必要なのは，ほとんどの場合レコードまたはセグメント中の1つまたは2つ程度のエレメントであることが多いからである。

データ・ベースへのアクセスを制限する方法として，もう1つ別に，DATBAS モジュールを経由して行なうことができる。この方法は，例えばあるユーザーに，特定のエレメントまたはデータ・セットの"検索"は許可するが"更新"は許さない，ような場合に便利である。この場合，DATBAS に"パスワード"を定義して，どのユーザーにどのようなオペレーション……読み，書き，追加，削除……を許可するかを定める。そしてその制限は，どのデータ・ベース，データ・セット，エレメントについてでも設けることができる。

データの保護 ― 予期できない破壊の防止

前に述べたとおり，従来の順次処理システムでは，アプリケーションごとにファイルがあり，ファイルを共用することは稀であり，従って"データの保護"ということも大して問題とならなかった，というよりも問題外であった。あるアプリケーションがデータを破壊したとしても，その影響を受けるのは通常，当該アプリケーションのみである。

"データ・ベース"システムになると，多くのアプリケーションが同じファイルを共用し，依存しているために，データの保護ということが重要な問題になってくる。あるアプリケーションが，あるデータを破壊した場合，その影響は多くの他のアプリケーションに及ぶ。

TOTAL は予期できない破壊からデータ・ベースを保護するために，次のような方法を用意している。

＊ ユーザーのデータ・ベース CALL命令によって，TOTAL が要求された機能を遂行しようとしたとき，故障または異常状態になった場合には，TOTAL は要求された機能を実行しないで，ユーザーに異常状態の原因を知らせる4文字のステイタス・コードを返す。そのいくつかを例示する

DUPM …… マスター・データ・セットへレコードを重複（DUPlicate）して追加しようとしている

IPAR …… パラメータ（PARameter）リストが間違（Invalid）っている

FULL …… 記憶装置容量が既に限界に達している

IMDL …… マスター・レコード削除命令は間違っている。これはあるマスターレコードを削除しようとしたが，まだこれにリンクしているバリアブル・レコードが残っている場合である

TOTAL はさらにいくつかのデータ・ベース破壊防止法を用意している。

＊ ユーザーはマスター・レコード内の"ルート"エレメントや，マスターとバリアブル・レコード内の"リンケージ・パス"エレメントをアクセスすることができない。これらの特別なエレメントにはポインターが入っていて，TOTAL はこれをマスター・データ・セットのシノニムのチェインを作るため，または，マスター・データ・セットから一連のバリアブル・レコードへのリンクをとるために使っている。図８－３参照。これらのエレメントは決してユーザー・プログラムへ渡されないので，ユーザーがその中にあるポインターを壊わすということは起り得ない。

＊ すべての TOTAL のポインターは最初と最後および直前と直後（図８－３では FRST と LAST および PREV と NEXT ）の２つの方向をもっている。従ってもし片方の方向が予期せぬ破壊を受けた場合，TOTAL はもう一方の方向から，修復することができる。TOTAL が壊れたポインターを自動的に直せない場合を除いて，ユーザーはそのような破壊について何も気がつかないうちに TOTAL はこのような破壊があれば自動的に直してしまう。

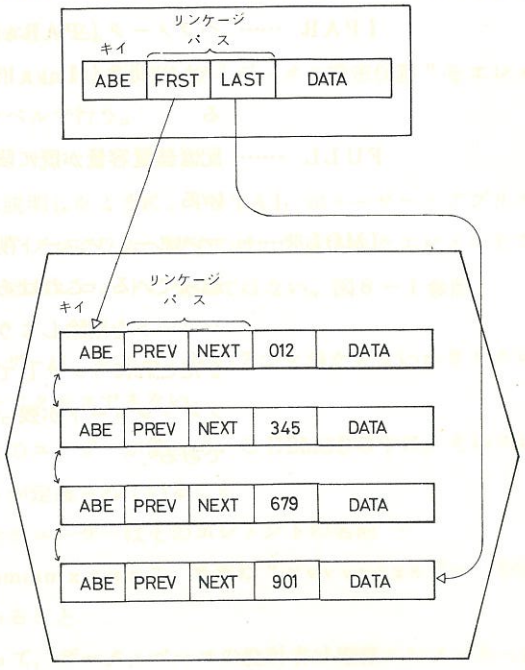


図８－３ リンケージ・パス

＊ TOTAL は複数のユーザー・プログラムが，同時に同じデータ・ベースを更新しようとしたときに起る破壊を防ぐ。このやり方はさらに複雑であるので，次節で詳しく解説する。

データの保護 ― 同時更新からの保護

ここでは複数のユーザー・プログラムが，同時にデータ・ベースを更新しようとするとき，データ・ベースを保護する TOTAL の機能について解説する。TOTAL４と TOTAL ５／６ は共に同時更新保護機能を備えているが，TOTAL ５／６ は特に複数のプログラムが同時にデータ・ベースにアクセスする多重処理向に設計されており，TOTAL４ は１タスクの単純処理向設計である。従って多くのタスクが同時にデータ・ベースにアクセスする環境では TOTAL５／６ の保護機能の方が有効である。

TOTAL ４

DOS および OS の TOTAL４ には"TOTAL"，"MPTOT"，"QUEST"の３つのデータ・アクセス・モジュールがある。図８－４参照。

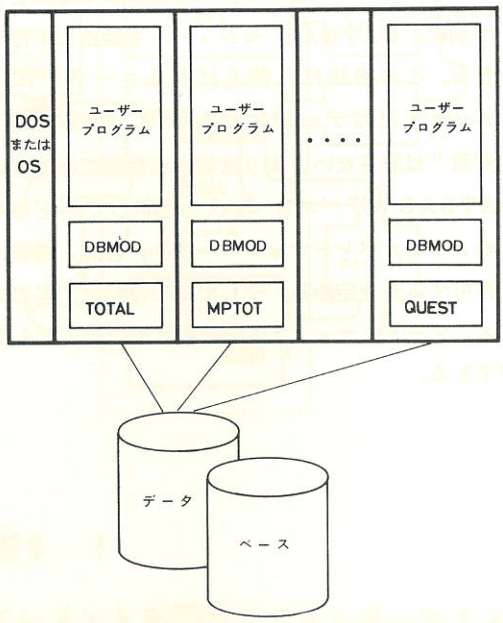


図８－４ TOTAL ４

これらのモジュールがデータ・ベースに対して働きかける機能は異っている。

TOTAL	レコードの追加
	レコードの削除
	エレメントの書き込み
	エレメントの読み込み
MPTOT	エレメントの書き込み
	エレメントの読み込み
QUEST	エレメントの読み込み

これらのモジュールは，いくつかの異ったアプリケーション・プログラムとともにコア内にあり，同時並行的に稼動していることがある。しかし，コア内においてこれらのモジュールが相互に連絡しあう方法はない。

それでは，これらの独立した複数のモジュールが，同時に1つのレコードを更新するのを防ぐのは何であろうか。

これを知るために，それぞれの更新オペレーションの内容を調べてみよう。

追加と削除

各アプリケーション・プログラムは，TOTAL データ・セットに対して何かの命令を実行するときはその前に，次の形式でDBCL CALL を指示しなければならない。

```
CALL 'DATBAS' USING {'OPENM'},
                    {'OPENV'}
     STATUS,
     FILE,
     'END.'.
```

このコマンドにより，"FILE" パラメータで指定された1つのデータ・セットをオープンする。

TOTAL データ・アクセス・モジュールはデータ・セットをオープンするとき，そのデータ・セットに対し"IEOJロック"をセットする。このロックは次に述べるようにデータ・セットの中に物理的に設定される。

＊ マスター・データ・セットの場合，データ・セットの最終ブロック中の最終レコードの5バイト目に"IEOJ" ロック（'X' マーク）がセットされる。

＊ バリアブル・データ・セットの場合，IEOJ ロックはデータ・セットの第1ブロックの中の最初のレコードの21バイト目にセットされる。

ユーザーはデータ・セットの処理を終了すると，データ・セットをクローズするDBCL コマンドを出す。

```
CALL 'DATBAS' USING {'CLOSM'},
                    {'CLOSV'}
     STATUS,
     FILE,
     'END.'.
```

TOTAL データ・アクセス・モジュールがデータ・セットをクローズする場合は，IEOJ ロックのバイトにブランク（16進数'40'）を移しデータ・セットを"アンロック"する。

IEOJ ロックは，TOTAL の別のコピーが同じデータ・セットをオープンすることを防ぐ。別のTOTAL がデータ・セットをオープンしようとして，IEOJ ロックが"X"にセットされていると，ユーザー・プログラムに対し"IEOJ" というステイタスを戻す。

TOTAL はこのロックのセットやチェックをする唯一のデータ・アクセス・モジュールである。MPTOT や QUEST は，IEOJ ロックのセットやチェックを行わない。

TOTAL は，かつ，データ・セットに対しレコードの追加，削除の可能な唯一のモジュールでもあるから，IEOJ ロックは，同一データ・セットに対し複数のパーティションから同時に追加，削除が行なわれることを防ぐ効果をもつということができる。

書き込み

DBDL ステートメントのBLOCK－HOLD＝YES（オプション）は，TOTAL とMPTOT間の重複更新を防止する。

＊ TOTAL は1データ・セットにつき，1レコードをホールドすることができ，それは当該レコードをデータ・セットに書き込みを終るか，またはそのデ

ータ・セットから他のレコードを読み込むまで続く。

＊ MPTOT は1データ・ベースにつき，1レコードをホールドすることができ，それは当該レコードの更新が終るか，または他のレコードをデータ・ベースから読み込むまで続く。

TOTAL やMPTOT によりレコードがホールドされている間は，他の TOTAL やMPTOT のコピーはこのレコードをアクセスできない。

エレメントの書き込みができるのは，MPTOT と TOTALのみであり，この機能はレコードが複数タスクにより，同時に更新されるのを防ぐのに役立つ。

QUEST の場合は，レコードをホールドせず，TOTAL やMPTOT によりホールドされたレコードを読むことができる。

3つのデータ・アクセス・モジュールとその機能，IEOJ ロックおよびレコードのホールドの関係をまとめると次のとおりである。

TOTAL 4 同時更新からの保護

データアクセスモジュール	機能				IEOJロック	レコードのホールド
	読み込み	書き込み	追加	削除		
TOTAL	○	○	○	○	Yes	Yes
MPTOT	○	○			No	Yes
QUEST	○				No	No

TOTAL 5/6

TOTAL 5／6 は，コンピュータに入っているすべての処理タスクが，データ・ベースに対しアクセスするのを総括的に制御するプロセッサーである。図8－5参照。従って，TOTAL 5／6 も，同時更新の防止に関して，TOTAL 4と同じ機能をもっている。しかしマルチ・プログラムやマルチ・タスク状況下では，レスポンス・タイムおよびスループットの面で，TOTAL4 より一段と優れている。

TOTAL 5／6 はデータ・ベースに関して総括プロセッサーとして機能するので，各タスクのデータ・ベースへのアクセスを"スケジュール"することができる。第7章で述べたとおり，TOTAL 5／6 は，そのインターナル・タスク・テーブルを用いて，データ・ベースに対するアクセスを"シングル・スレッド"にスケジュールする。

TOTAL 5／6 は，TOTAL4 と同様，データ・セットをオープンする時はロックし，クローズする時はアンロックする。しかし TOTAL4 とは異り，このロックは他の TOTAL 5／6 のタスクがデータ・セットにレコードを追加・削除するのを防げない。

TOTAL 5／6 は，DBDL 中に BLOCK-HOLD ステートメントを必要としない。もし，このステートメントが存在しても TOTAL 5／6 では無視される。TOTAL 5／6 は，あるレコードの中の"エレメント"だけを論理的にホールドし，ブロック，データ・セットまたはリンケージをホールドすることはない。

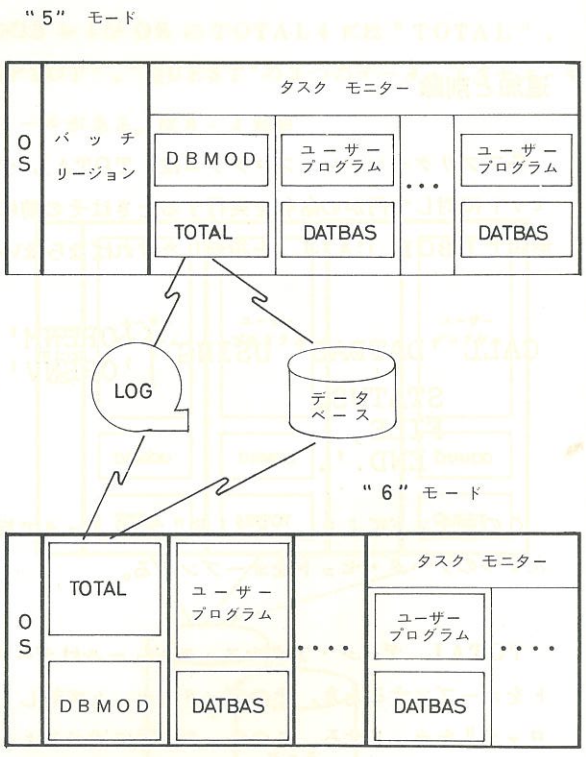


図8－5 TOTAL 5/6

従って逆にTOTAL 5／6の場合は，レコードを削除したり，そのエレメントに書き込んだりするため，レコードを"ホールド"する必要があるときは，ユーザー・プログラムでこれを行わなければならない。レコードをホールドするには，ユーザーは，CALL コマンドを使用して，当該レコードのREAD 命令（READM， READV， READR，READD）を出し，その最終パラメータは"END." とする。もし単に検索の目的――すなわち，レコードの書き込みや削除を伴なわない場合――のためにレコードを読み込む場合は，READ 命令の最終パラメータは"END." ではなく"RLSE"を使うべきである。RLSE が用いられていると，そのレコードまたはエレメントはホールドされない。

TOTAL 5／6 の場合READ命令によって1つのタスクは1データ・セットにつき，1レコード……当該レコードの1つまたは複数のエレメント……をホールドする。ユーザー・タスクが"END." パラメータを用いてレコードをREAD すると，次の条件のいずれか1つが成立するまで，そのタスクが当該レコードの当該エレメントに対するアクセスを専有する。

＊ そのタスクが，同じデータ・セットから別のレコードをアクセスするCALL 命令を出した場合。

＊ そのタスクが既に専有しているレコードに対して"RLSE"パラメータで終るREAD 命令を出した場合。

＊ "FREEX" コマンドを出すと，そのタスクが専有しているすべてのレコードまたはエレメントが解放される。

＊ TOTAL 5／6 はタスクのレコード・ホールドを自動的に終らせることができる。これは，あるタスクが余り長い時間レコードまたはエレメントをホールドし，そのため他のタスクがそのレコードをアクセスしようとして，立ち往生してしまう場合に起こる。TOTALは，このような場合，すなわち，ホールドされているレコードまたはエレメントをアクセスしたタスクがインターナル・タスク・テーブルにエントリされ，TOTAL のポーリングを1000回待っても，まだ当該レコードまたはエレメントが解除されていない時は1001回のポーリングでTOTAL は当該レコードまたはエレメントのホールドを解き，このタスクにレコードまたはエレメントを渡すのである（第7章参照）。

あるタスクが，"END." 付READ 命令でホールドしていないレコードに対して，それを削除したり，そのレコードに書き込もうとすると，TOTAL 5／6 は"NHLD"というステイタス・コードを返して，当該レコードまたはエレメントがまだホールドされていないことを示す。また，タスクがエレメントを書き込んだり，レコードを削除しようとするとき，このタスクの当該エレメントまたはレコードに対するホールドが解かれてしまっている場合にはTOTAL 5／6 はステイタス・コード"REPC" を返す。"NHLD" も"REPC" コードもともに，ユーザーはCALL ステートメントの最後に"END." 付のREAD 命令を出してから，WRITE やDELETE をやり直さなければならないことを示すものである。

TOTAL 4 とTOTAL 5／6の併用

OS TOTAL4 とOS TOTAL 5／6 は，同一のデータ・ベースに対し併用して同時処理ができる。この場合の同時更新の防止に関しては以下の点に注意する必要がある。

1. TOTAL 4 とTOTAL 5／6 が併用され，同時に同一データ・ベースをアクセスすると，データ・セット（IEOJ）ロックにより防止される。TOTAL 4 またはTOTAL 5／6 のいずれか一方がデータ・セットをオープンすると，他方はそのデータ・セットをアクセスできない。従って，TOTAL 5／6 のタスクは，TOTAL 4 によりオープンされているデータ・セットにアクセスできず，実行効率の面からみて，同一データ・ベースに対しTOTAL4 とTOTAL 5／6を同時に併用することは勧められない。
2. 同一データ・ベースに対しMPTOT とTOTAL 5／6 を併用する場合は，
   a）MPTOT はIEOJ データ・セット・ロックを無視する。
   b）BLOCK−HOLD 機能はMPTOT にあるが，TOTAL 5／6 にはない。

   従って保護機能が働かない。

データ・ベースの復旧

TOTAL のみならず，どのようなデータ・ベース・マネジメント・システムにおいても，データ・ベースが壊われるということを避けることはできない。どのようなソフトウェア・テクニックを使っても，火災，ディスク・ヘッドの損傷，電源の切断などを防ぐことはできない。ユーザー・アプリケーション・プログラムの誤りによるデータ・ベースの破壊も，DBMS は防ぐことができない。

このようなことに対して，TOTAL は次の構造と機能をもつ。

＊ データ・ベースの破壊の範囲を限定するモジュール別データ・ベース構造

＊ データ・ベースの破壊を速やかに復旧する機能

TOTAL のこれらの面について以下に述べる。

データのモジュール構造

データ・ベースの破壊からの復旧……リストア（Restore）……という観点から，TOTAL の最も重要な面の1つは，そのデータ・ベース構成がモジュール構造になっているということである。TOTAL では破壊があっても，常にその程度はおよそ1つのデータ・セットまたは数個のデータ・セットにとどまるのである。図8－6参照。

TOTAL データ・セットが壊れたとき，バックアップ・コピーまたはデータ・ベース・ログよりリストアする。ここで重要なことはデータ・ベースをリストアすると，そのレコードの位置（すなわちアドレス）は変化するがレコードのキイは変化しないということである。それ故に，

＊ バリアブル・データ・セットをリストアする場合，そのバリアブル・データ・セットへのリンケージ・パスを持っているすべてのマスター・データ・セットもまたリストアしなければならない。

＊ しかし，マスター・データ・セットをリストアする場合，そのレコードのキイは変化しない。

バリアブル・データ・セットはマスター・データ・セットへリンクするためにこのキイを使うのであるから，マスター・データ・セットをリストアしたときバリアブル・データ・セットをリストアする必要はない。

ダンプとリストア

データ・ベースが壊されたとき，リカバリーの最も単純な方法は，最新のバックアップ用コピーで壊されたデータ・セットをリストアし，次にコピーが作られた以後にデータ・セットを更新したすべてのプログラムを再処理することである。この方法は当然にバックアップ用コピーが事故の発生する前に作られていることを前提としている。そしてこれらのコピーはデータ・セットをテープまたはその他の記憶媒体にダンプすることによって作られる。標準的なダンプ／リストア用の汎用プログラム，例えば IBM の IEHDASDR や㈱アシストの FDR & DSF はいずれもバックアップ用コピーを作り，このコピーから壊されたデータ・セットを復旧することに利用されている。図8－7参照。

このテクニックは非常に単純ではあるが，費用対効果の面で最も優れており，バッチ処理方式においては最も広く用いられているものである。

この方法は TOTAL 4 や TOTAL 5／6 のいずれの場合にも用いられる。TOTAL 4 にあってはこれが唯一の復旧の方法である。

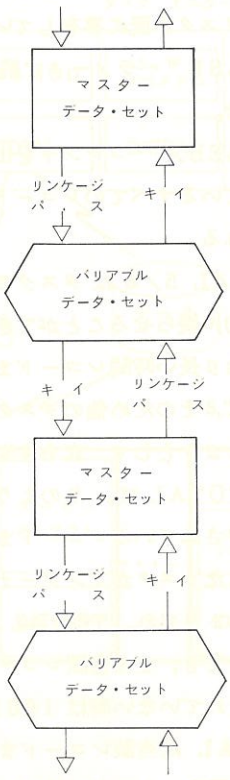


図8－6
リンケージ・パスとキイの役割

どの程度の頻度でデータ ベースのバックアップをとるべきかは次の２つの条件を比較検討して判断することになるだろう。

＊ データ・ベースをバックアップするためにどれくらい時間を必要とするか

＊ 最新のバックアップ・コピーをデータ・ベースにリストアし，そこから必要なすべてのトランザクションを再処理するためにどれくらいの時間が必要か

さらに考えねばならない他の要因として次のものが考えられる。

＊ もし，データ・ベースが壊れたら，どの程度速やかにそれを復旧する必要があるか。このことはデータ・ベースの性質にもよるし，そのデータ・ベースに依存しているアプリケーションの性質にもよる。

＊ バックアップはコンピュータが最も暇なときにとるよう計画することができる。復旧は予め計画する事ができない。なぜなら事故を予見することができないからである。従って，ユーザーはスケジュールにない復旧ジョブの実行が，コンピュータ・オペレーション全体に与える影響を考慮しなければならない。

＊ 最後に，どの程度の頻度でデータ・ベースがこわれるであろうか。このことはデータ・ベースが存在している環境にかかわりあいを持っている。

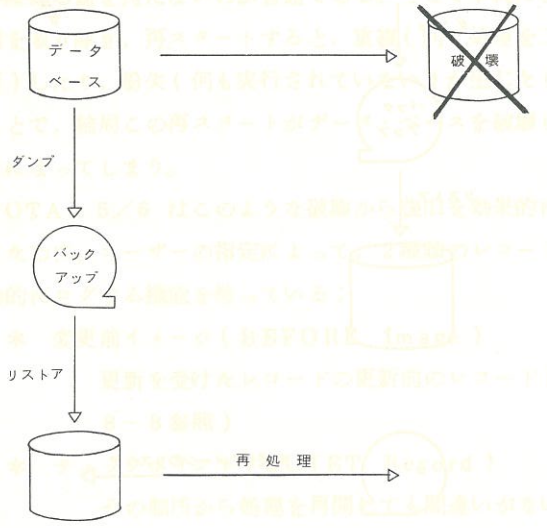


図８－７ ダンプとリストア

ログとリストア

ダンプ・リストアの方法はデータ・ベースがバッチ処理方式でのみ使用されている場合のデータ・ベースの復旧法として一般的に認められている方法である。もしデータ・ベースがオンライン・システムで使用されている場合はどうか？

バックアップ用コピーが早朝オンライン・システム始動前に作成されていたと仮定しよう。そして事故……データ・ベース破壊……がオンライン・システム稼動後６時間たった時点で発生したとする。もし前述のダンプ・リストア方式をとったとするならば，この場合復旧に要する時間はリストア時間＋再処理時間……最大６時間……となって，この日はオンライン処理はダメということになる。

当然のことながら，オンライン環境下では万一事故が起った場合に再処理しなければならないトランザクションの数を極力少なくする配慮が行なわれることは明らかである。その場合，１５分ごとにオンラインを停止してファイルのコピーを作るというのは必ずしも正解とはいえないが，１５分ごとにデータ・ベースの"ステイタス"を確認するという方式は合理的といえるであろう。それにはどうすればよいか？

TOTAL の持つ重要な特長の1つは IBM のアクセス・メソッド……すなわち"DIRECT" アクセス法……にある。ダイレクト・アクセスとは各ファイルのレコードがそれぞれ固有で変更されることのない番地を持っていることを意味する。さらにもう1つ，ダイレクト・アクセスは次の場合すべて READ/WRITE を行なう：

* あるレコードを新たに追加する場合，ブランク・レコードが読み込まれ，次にデータの入ったレコードが書き込まれる。
* あるレコードが削除される場合，データの入ったレコードが読み込まれ，次にブランク・レコードが書き込まれる。
* あるレコードが更新される場合 （これには"WRITM" と "WRITV"がある）あるデータの入ったレコードが読み込まれ，次に別のデータが入っているレコードが書き込まれる。図8－8参照。

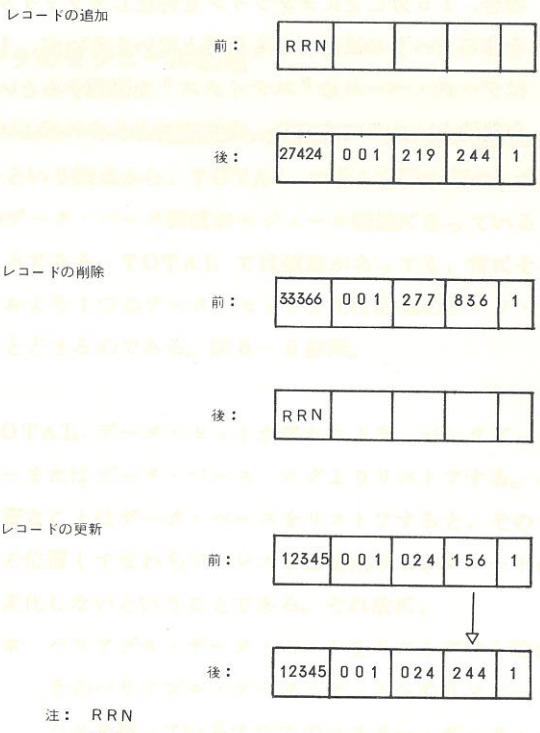


図8－8 ダイレクト・アクセス

上記いずれの場合にも書き込まれるレコードは，読み込まれたレコードがもと書かれていた番地に返される。このダイレクト・アクセス法の性質が，オンライン下でデータ・ベースを復旧する TOTAL の復旧手法の基礎になっている。

変更後イメージ

そこでもう1度オンライン開始前にバックアップ・コピーを作り，システム稼動後6時間経過したとき，ディスク装置のヘッドが壊れるとか，別の事故で，ともかくデータ・セットのいくつかが，物理的に破壊された場合を想定しよう。データ・セットは物理的に破壊されているので，アクセスはもちろんできず，最新のコピーから復旧するほかはない。データ・セットを復旧し，コピー作成時点以後のアプリケーションを再処理すれば，既に述べた通り6時間かかってしまう。

TOTAL 5／6 はこの再処理時間を劇的に短縮してくれる。TOTAL 5／6 を使うときに，あるパラメータを指定しておくと，TOTAL は自動的にログ用ファイル……テープまたはディスク……に変更（追加，削除，更新）したすべてのレコードの変更後イメージ（AFTER Image）を記録する。さらに TOTAL は，ログ・ファイルを読み，そこに記録されたすべての変更後イメージ・レコードをデータ・ベースに直接書き込む汎用ルーチンを持っている。図8－9参照

この TOTAL のログ機能を使えば，復旧に要する時間は，オンライン稼動中（上記例では6時間）の再処理時間は，変更されたレコードの変更後イメージをデータ・ベースに読み

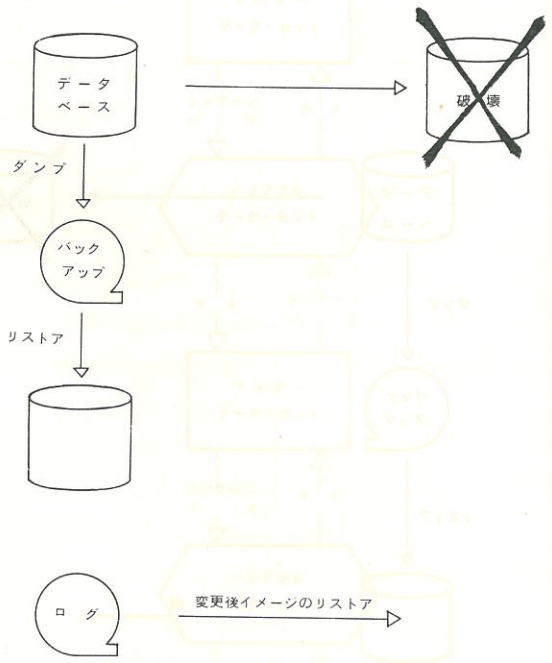


図8－9
データ・ベースの復旧……バックアップ・テープから

込むだけの時間に短縮することができる。通常この短縮の程度は極めて大きいはずであるが，復旧にどれ位時間がかかるかは，バックアップ・コピーが作られてから事故が発生するまでの間にデータ・ベースに加えられた"変更"の量に依存することは当然である。

変更前イメージとQUIETレコード

例えばディスク・ヘッド損傷すなわち，物理的にデータが破壊されるというケースは今日では滅多に起らないと思われるが，次のようなケースは稀ではない：

* 電源故障
* プログラムの異常中断（Abend）
* OSの誤動作
* 端末から誤ったデータを書き込む
* プログラムの間違い

もちろん，このような場合にはデータの物理的破壊……読み出し不能……にはならないであろうけれども，データ・ベースの信頼性を傷つけることは明らかである。例えば，プログラムに間違いがあったり，端末から間違ったデータを送り込んだ場合はレコードが正しく追加，削除，更新されたとはいえないことが多い。また電源が故障した場合には，プログラムが異常中断したり，オペレーティング・システムが誤動作し，データ・ベースのデータは破壊されないとしても，その時点でどのレコードは更新されたか，更新されるべきレコードは正しく更新が終了しているかどうか，などについてユーザーは知る術を持たないのが普通である。そこで単純に事故原因を取り除き，再スタートすると，重複（同じ処理を二度実行）したり，紛失（何も実行されていない）が生じたりすることで，結局この再スタートがデータ・ベースを破壊したことになってしまう。

TOTAL 5／6 はこのような破壊から復旧を効果的に行なうために，ユーザーの指定によって，2種類のレコードを自動的にログする機能を持っている：

* 変更前イメージ（BEFORE Image）
  更新を受けたレコードの更新前のレコード（図8－8参照）
* チェックポイント（QUIET Record）
  その個所から処理を再開しても間違いがないことを示すレコード

TOTAL はバッファ内でデータ更新が行なわれると，その都度，常にログ・ファイルにログ・レコードを書き込む。従って，アプリケーション・プログラムとTOTALのバッファとログ記録とは常に同期している。然しながらTOTALのバッファ・ロジックは図8－10に示すとおり，バッファ内にある更新されたレコードをデスクへ物理的に書き込むのは，当該バッファに新しい別のレコードを読み込む必要が起るまで実行しない。図8－10参照。そのためにディスク上のデータ・ベース自体はアプリケーション・プログラム，TOTALのバッファ，ログ・ファイルと通常一致していない。

"QUIET" はバッファ内の更新されたすべてのレコードを物理的にディスクに書き込ませる機能をもち，これによってデータ・ベースとアプリケーション・プログラム，TOTALのバッファ，ログ・ファイルの一致を実現することができる。このようにして物理的書き込みが行なわれたとき，TOTALは，特殊なQUIET レコードをログ・ファイルに書く。

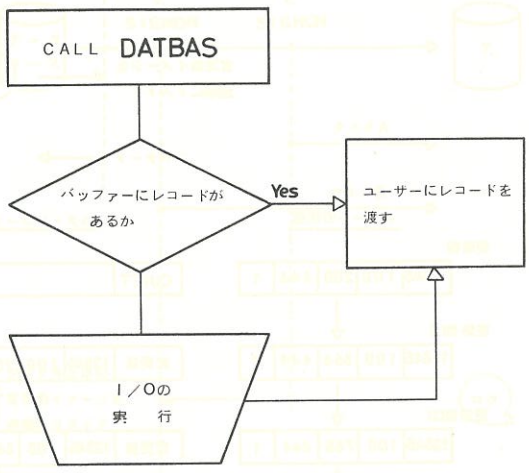


図8－10 バッファ・ロジック

変更前イメージと"QUIET"レコードはTOTALデータ・ベースを復旧する場合最も多く利用される手法の骨格をなすものである。図8－11はその使い方を例示している。すなわちオンラインをスタートしてから数時間たった時点で電源が切れたとする。データを更新したトランザクションについて紛失も重複処理もすることもなく，システムを復旧するために，まず第1にデータ・ベースを，最後のチェック・ポイント（すなわち最後のQUIETレコード）をとった時点の状態に復旧する。この場合仮に最後にQUIETをとってから，事故が発生するまでにデータ・ベースに更新を加えたトランザクションが3個であったとする。それは例えば図8－11の左下に示す内容で，更新の都度，変更前イメージがログ・ファイルに記録されている（図右側）。そこで最終チェックポイント"QUIET"時点の状態にデータ・ベースを戻すにはどうすればよいかは極めて自明のことで，ログ記録を終りの方から逆読みして ……逆順に…… この3つの変更前イメージのレコードをデータ・ベースに書き込めばよい。そして，まさにこの通りのことを実行するプログラムをTOTALは用意している。

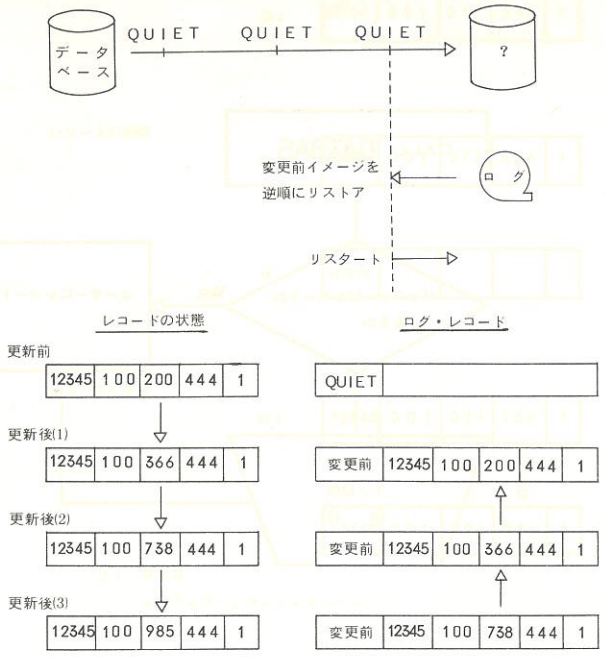


図8－11 データベースの復旧(1)

このように"QUIET"ポイントからオンライン・システムを再開する限り，トランザクションの紛失，重複は全く心配することはない。さらに復旧に要する時間の節約も測り知れぬと思われるけれども，ここで強調したいことは，データ・ベースの破壊がオンライン始動後6分後であったか，6時間後であったか，それとも16時間後であったかということには関係なく，最後のチェックポイント（"QUIET"）以後の変更前イメージをデータ・ベースに書くだけで，システムを再開することができるという点である。

SIGNON，SIGNOFFレコード

オンラインを始動してから数時間経過したとき，データ・ベースを更新するアプリケーションの中の1つにプログラムのミスがあって，間違った追加，削除，または変更を加えてしまった場合を考えてみよう。このようなトラブルを起したタスク── タスクA ──のためにデータ・ベースは既に破壊されてしまった。これを復旧するためには，次の2つのことのうちいずれかを実行しなければならない。

1. もし，タスクAは間違った更新を行ったけれども，他のタスクには別段の影響は無かった場合には，タスクAが追加，削除，変更したレコードの変更前イメージをデータ・ベースに書き込む
2. タスクAの誤処理の結果が，他のタスクの処理に影響している場合には，その復旧は複雑になる。例えばタスクAはある部品在庫を実際には100あるのに誤って0にしてしまった。別のタスクはこの部品在庫の記録を読んで，在庫量0であるため500の発注書を発行した。この後者のタスクのプログラムには間違いはないけれども，タスクAによって作られた間違ったデータを基にしているために，次のタスクで作られた発注書は正しくないし，データ・ベースから間違った記録も訂正されなければならない。

   このような場合，タスクAによって更新されたすべてのレコードの変更前イメージをデータ・ベースに書くと共に，このタスクAの誤りによって影響を受けたと思われる他のすべてのタスクのレコードの変更前イメージを残らずデータ・ベースに書き込まなければならない。

上記いずれの場合でも，TOTALを使っている場合には，ユーザーはあるパラメータを指定してTOTALにどのタスクがいつ"実行開始の合図（サイン・オン）"をしたか，"実行終了の合図（サイン・オフ）"をしたかをログ記録にとり，復旧を容易にすることができる。このログ記録を"SIGNON"，"SIGNOFF"と呼ぶ。

図８－１２参照。そこで，データ・ベースを復旧するにはログ記録を終りの方から逆順に読み，タスクＡおよびタスクＡの更新によって影響を受けたと思われるタスクの変更前イメージを，タスクＡの＂ＳＩＧＮＯＮ＂レコードより古い＂ＱＵＩＥＴ＂レコード（図では＂ＳＩＧＮＯＮ＂の左側で最新の＂ＱＵＩＥＴ＂レコード）のところまでデータ・ベースに書き込んでいく。

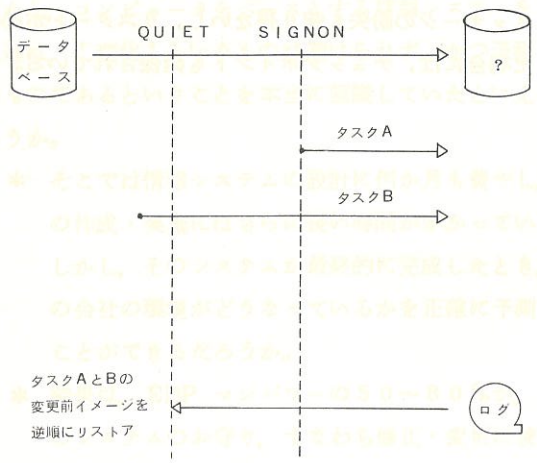


図８－12 データベースの復旧(2)

QUIET，QMARK 命令

既に述べたとおり，ユーザーはＴＯＴＡＬに＂ＱＵＩＥＴ＂レコードを自動的にログ記録に書かせることができる。それはＴＯＴＡＬがスタートするときに，あるパラメータを指定することによって実行される。そのパラメータは更新（追加，削除，変更）がいくつ行なわれるごとに＂ＱＵＩＥＴ＂を記録するかを指定するものである。

＂ＱＵＩＥＴ＂は前述のとおり一種のチェックポイントであり，その時点から安心してデータ・ベースの処理を再開してもよいことを示すものである。しかし，ユーザーのアプリケーション・プログラム，またはオンラインの制御プログラムにチェックポイントからの再処理を実行する機能が用意されていなければ，チェックポイントは無意味になってしまう。（図８－１２でタスクＢが＂ＱＵＩＥＴ＂時点から――すなわちプログラムの途中から――リスタートできない場合）

このような場合にユーザーが必要とする目印は：

＊ 安心して処理を再開することができる目印……すなわちデータ・ベースが，アプリケーション・プログラム，ＴＯＴＡＬのバッファ，ＴＯＴＡＬのログ記録と一致している時点のしるし

＊ 処理を再開することが可能な目印……すなわち，アプリケーション・プログラム並びにもし必要ならばオンライン制御プログラムがリスタートできる時点を示すしるし

ＴＯＴＡＬはこのような場合に復旧やリスタートを効率よく実行できるためのＤＢＣＬ命令を特別に用意している。それはＱＵＩＥＴ命令とＱＭＡＲＫ命令で，共にＴＯＴＡＬにデータ・ベースのチェックポイント……この命令が出たときバッファ内容をすべてディスクに書き出す……を作らせる。

図８－13 参照。

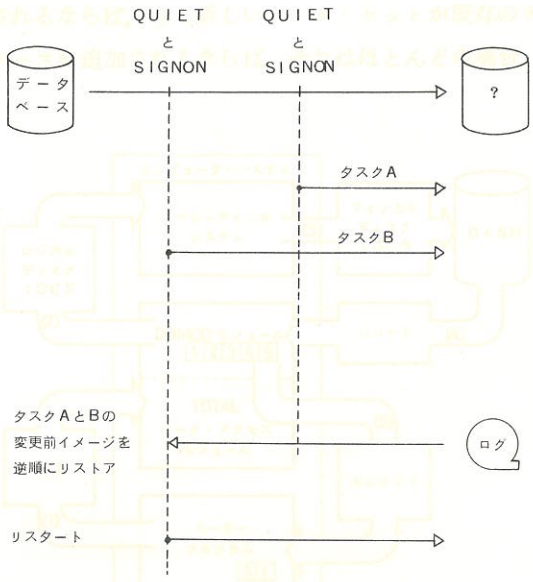


図８－13 データベースの復旧(3)

この命令はTOTALを呼び出したアプリケーション・プログラムから次のDBCLステートメントのいずれかを出すことによって行なわれる：

```
CALL 'DATBAS' USING 'QUIET',
     STATUS,
     COUNT,
     'END.'.

CALL 'DATBAS' USING 'QMARK',
     STATUS,
     COUNT,
     AREA,
     'END.'.
```

"COUNT"パラメータはQUIETレコードを更新データの数が何個ごとにつけるかを指定するもので，TOTALがスタートした時に既に指定されている数を変更する時に用いる。上記2つの命令の相違点は，QMARK命令には"AREA"パラメータがあり，これはユーザーが復旧やリスタートを行なうときに使うログ・ファイルに記録したいデータがどこにあるかを指定するものである。

アプリケーション・プログラムがTOTALをサイン・オン，サイン・オフする時には，通常"QUIET"命令も出すとよい。

オンラインでCICSのようなログ機能に極めて弱い制御プログラム下で，トランザクション処理型のアプリケーションを行なう場合，QMARK命令は偉力を発揮する。すなわち端末から新しい入力メッセージを読むたびにアプリケーション・プログラムはQMARK命令を出し"AREA"パラメータを指定することによってこのメッセージをTOTALのログ・ファイルに書き込むことができる。こうしておけば，メッセージの紛失も起り得ないし，リスタートが必要となった場合には，チェックポイントも確保されている。

第９章　データ・ベースの変化

変化に対する適応性

我々は激動の時代に生きている。我々が直面する変化のほとんどは避けられないばかりか，予測することもできない。

しかし，コンピュータをベースとする情報システムを作った人達は，＂変化＂というものは避けられず，かつ予測できないものであるということを本当に認識していたといえるであろうか。

＊ そこでは情報システムの設計に何か月も費やし，その作成・実施にはさらに長い時間がかかっている。しかし，そのシステムが最終的に完成したとき，その会社の環境がどうなっているかを正確に予測することができるだろうか。

＊ 結果は，EDP マンパワーの５０～８０％が，過去のシステムのお守り，すなわち修正・変更に使われてしまうことになる。

TOTALはこのようなムダを取り除く手段を提供するものである。この章で説明するように，TOTALのデータ・ベースは会社が成長し，情報の要求が変化するのに合せて成長し，変化することができる。そして設計，メンテナンスおよびコンバージョンにかかる費用を大幅に削減することができる。なぜなら，

＊ TOTALはデータとそのデータをアクセスするアプリケーション・プログラム間に，高度の独立性を持たせている。

＊ TOTALのデータ・ベースは高度にモジュール化されている。

データの独立性

TOTALデータ・ベースが変化に対し，高度の適応性を持つもう１つの理由は，データとそのデータをアクセスするアプリケーション・プログラム間の高度の独立性である。このデータの独立性によりデータ・ベースの変化が，ほとんどの場合，既存のアプリケーション・プログラムへは影響を与えない。

データ・ベース・コマンド言語（DBCL）を使って，アプリケーション・プログラムはTOTALに CALL をかけ，必要なエレメントだけを要求し，必要なエレメントだけを受取る。TOTALはレコード全体を読むのではあるが，アプリケーション・プログラムへは，DBCL CALLステートメントで指定したエレメントのみを渡すのである。図９－１参照。

モジュール構造

TOTALデータ・ベースが変化に対して，高度な適応性を持っている第１の理由は，データ・ベースが実際にモジュール化されているからである。このモジュール構造により，データ・ベース全体の作り直しや再ロードなどをしなくても，データ・ベースの部分的変更が可能になる。

もし新しいエレメントが既存のレコードに追加されるならば，もし新しいリンケージ・パスが既存のデータ・セットに追加されるならば，もし新しいデータ・セットが既存のデータ・ベースに追加されるならば，それはほとんどの場合，１

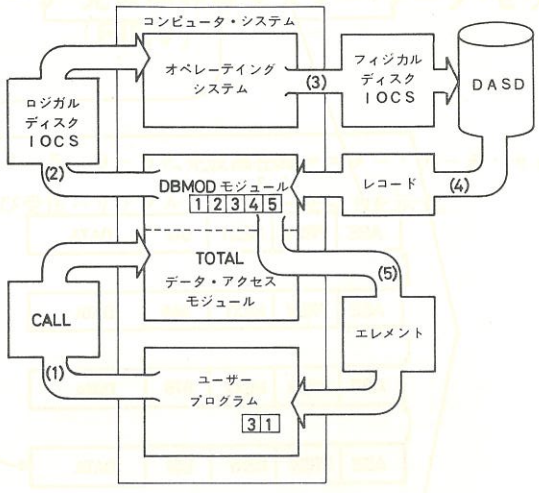


図９－１

エレメントを要求してから受け取るまで

つまたは少数のデータ・セットを変更するにとどまる。すなわち，1つまたは少数のデータ・セットを再定義し，再フォーマッティングし，再ロードする必要がある。データ・セットを再ロードするとき，そのレコードの位置 … つまり，アドレス … が変化する。しかし，そのレコードのキイは変化しない。すなわち，データ・ベースを再ロードしても，客先番号とか商品番号のような論理的情報 … データ … は変化しない。

TOTALはマスター・データ・セットからバリアブル・データ・セットへリンクするのにリンケージ・パス（すなわち，アドレス）を使い，バリアブル・データ・セットからマスター・データ・セットへリンクするのにキイを使う。図9－2参照。それ故，データ・セットの再ロードの影響は，

＊ マスター・データ・セットを再ロードするとき，他のいかなるデータ・セットもその影響を受けない。

＊ バリアブル・データ・セットを再ロードするとき，影響を受けるデータ・セットは，そのデータ・セットへのリンケージ・パスを持っているマスター・データ・セットだけである。それらのマスター・データ・セットだけを再ロードすればよい。

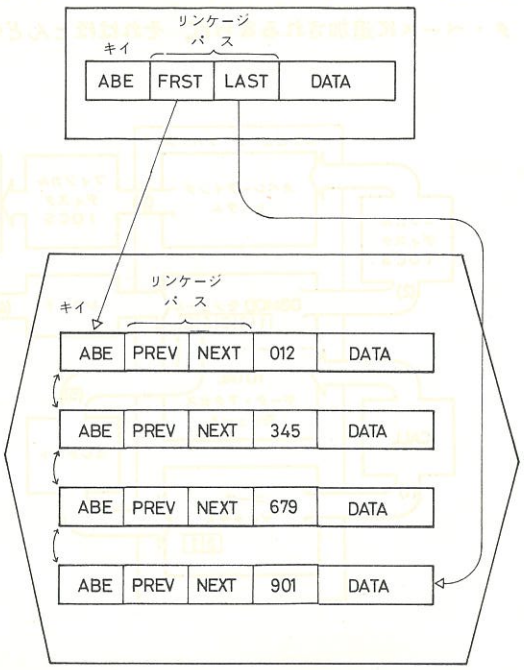


図9－2 リンケージ・パスとキイ

データ・ベースの成長と変化

最初のデータ・ベース

TOTALのモジュール構造とデータの独立性がいかに有効であるかを次の例で説明しよう。

システム・アナリストに，次のアプリケーション作成の仕事が与えられたとしよう。

＊ 受注処理

＊ 請求書発行

＊ 完成品の在庫管理

TOTALによって，これらのアプリケーションを，どのように実施に移したらよいか。分析の結果，これらのシステムが必要とするファイルは，得意先ファイル，受注ファイル，そして完成品（すなわち商品）在庫ファイルであると決定したとする。図9－3参照。

＊ 得意先および完成品在庫ファイルは，おそらくマスター・データ・セットであり，そのアクセスは得意先番号，商品番号などのはっきり区別できるキイで

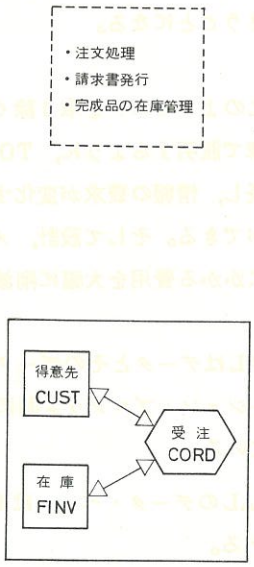


図9－3 最初のデータ・ベース

行う。これらのデータ・セットは四角で表わす。

＊ 受注ファイルに注文を記入するためには得意先番号を，また在庫管理を行なうためには商品番号をそれぞれのキイにして，アクセスできるようにする必要がある。この受注ファイルは，バリアブル・データ・セットとして六角形で表わす。

最後にそれぞれの必要とする関係を示すために二方向性の矢印で結ぶ。

＊ 特定の得意先からのすべての注文

＊ 特定の商品のすべての注文

得意先 マスター・データ・セット（CUST:customer）

エレメント	処理名	長さ
得意先番号	CUSTCTRL	6
得意先名	CUSTNAME	30
得意先住所	CUSTADDR	30
得意先所在都市	CUSTCTYS	20
得意先郵便番号	CUSTZIPC	5
特別信用条件	CUSTCRED	6

フォーマット

ルート	得意先番号キイ	受注リンケージ・パス	得意先名	得意先住所	得意先所在都市	得意先郵便番号	特別信用条件
8	6	8	30	30	20	5	6

定義

```
BEGIN-MASTER-DATA-SET:
DATA-SET-NAME=CUST
MASTER-DATA:
CUSTROOT=8
CUSTCTRL=6
CUSTLK01=8=CORD
CUSTNAME=30
CUSTADDR=30
CUSTCTYS=20
CUSTZIPC=5
CUSTCRED=6
END-DATA:
```

図9－4 得意先マスター・データ・セット(1)（CUST）

次に各データ・セットにどのデータをいれるかを定義する。

図9－4に得意先マスター・データ・セットを示す。アイテムをまとめ，エレメントごとに名前と長さを決める。アプリケーション・プログラムがデータ・ベースと情報のやり取りをするのは，このエレメント単位である。

完成品在庫 マスター・データ・セット（FINV:finished-goods inventory）

エレメント	処理名	長さ
品目番号	FINVCTRL	8
品目名	FINVDESC	30
所在倉庫	FINVWLOC	5
品目単位	FINVPRIC	5
手持在庫	FINVONHD	4
受注残	FINVORDR	4
発注点	FINVRORD	4
経済発注	FINVEORQ	4

フォーマット

ルート	品目番号キイ	受注リンケージ・パス	品目名	所在倉庫	品目単位	手持在庫	受注残	発注点	経済発注
8	8	8	30	5	5	4	4	4	4

定義

```
BEGIN-MASTER-DATA-SET:
DATA-SET-NAME=FINV
MASTER-DATA:
FINVROOT=8
FINVCTRL=8
FINVLK01=8=CORD
FINVDESC=30
FINVWLOC=5
FINVPRIC=5
FINVONHD=4
FINVORDR=4
FINVRORD=4
FINVEORQ=4
END-DATA:
```

図9－5 完成品在庫マスター・データ・セット（FINV）

図9－5，9－6に完成品在庫マスター・データ・セットおよび受注バリアブル・データ・セットを示す。

受注バリアブル・データ・セット（CORD: customer open order）

レコード・タイプ	エレメント	処理名	長さ
すべてのレコード	受注番号	CORDORNM	12
	ライン番号	CORDLINE	2
ヘッダー・レコード	得意先番号	CORDCUST	6
	受注日	CORDDATR	4
	出荷日	CORDDATS	4
	総受注額	CORDVALU	9
	受注数量	CORDTOTI	2
	受注条件	CORDTERM	10
コメント・レコード	コメント	CORDCOMT	55
品目レコード	品目番号	CORDFINV	8
	受注数量	CORDQTYX	4
	品目単価	CORDPRCE	8
	品目重量	CORDWHTX	4

フォーマット

←ベース・データ部→ ←――― 再定義　データ部 ―――→

HD	キイ 受注番号	ライン番号	キイ 得意先番号	リンケージ・パス 得意先	受注日	出荷日	総受注額	受注数量	受注条件	ブランク
2	12	2	6	8	4	4	9	2	10	12

CM	キイ 受注番号	ライン番号	コメント
2	12	2	55

IT	キイ 受注番号	ライン番号	キイ 品目番号	リンケージ・パス 完成品在庫	受注数量	品目単価	品目重量	ブランク
2	12	2	8	8	4	8	4	23

定　　義

```
BEGIN-VARIABLE-ENTRY-DATA-SET:
DATA-SET-NAME=CORD
BASE-DATA:
CORDCODE=2
CORDORDM=12
CORDLINE=2
CORDRDEF=55
RECORD-CODE=HD
CORDCUST=(6)=CUSTCTRL
CUSTLK01=(8)=CORDCUST
CORDDATR=(4)
CORDDATS=(4)
CORDVALU=(9)
CORDTOT1=(2)
CORDTERM=(10)
RECORD-CODE=CM
CORDCOMT=(55)
RECORD-CODE=IT
CORDFINV=(8)=FINVCTRL
FINVLK01=(8)=CORDFINV
CORDQTYX=(4)
CORDPRCE=(8)
CORDWHTX=(4)
END-DATA:
```

図9－6　受注バリアブル・データ・セット（CORD）

受注バリアブル・データ・セットは他の2つとはちがい，異る3つのフォーマットのレコードから成る。すなわち，コード付レコードである。まずすべてのレコードに表われるエレメントである受注番号とライン番号をベース・データ部に定義し，次に各再定義データ部に該当するエレメントを定義する。"DATA-SET-NAME＝CORD" はデータ・セット名を定義する。"BASE-DATA：" の次から，ベース・データ部の定義が始まる。

"RECORD-CODE＝HD" の次から，レコード・コード "HD" の再定義データ部の定義が始まる。"CORDCUST＝(6)＝CUSTCTRL" は，この "HD" コード付レコードと得意先レコード（CUST）を関連づけるキイを定義する。"CUSTLK01＝(8)＝CORDCUST" はTOTALがレコードを関連づけるために使うリンケージ・パスである。これは得意先データ・セット（CUST）を定義したときと同じ名前である。また，得意先番号と "CUSTLK01" リンケージ・パスがあるのは "HD" レコードのみであるから，このタイプのレコードのみが得意先レコードとリンクできる。同様に "CORDFINV＝(8)＝FINVCTRL" と "FINVLK-01＝(8)＝CORDFINV" は，"IT" コード付レコードと同じキイをもつ完成品在庫（FINV）レコードを関連づけるのに使われる。これらの3つのデータ・セットの物理的特性は，"END-DATA：" ステートメントの後に定義される。

DBDL ソース・ステートメントのコーディングが終ったらパンチして，TOTALのデータ・ベース作成プログラムDBGENに入力する。DBGENはデータ・ベース定義モジュール（DBMOD）を，アセンブラー・ソース・ステートメントの形で出力するので，このDBMODをアセンブルしライブラリーに登録する。図9－7を参照。これらの処理に要する時間は非常に短い。

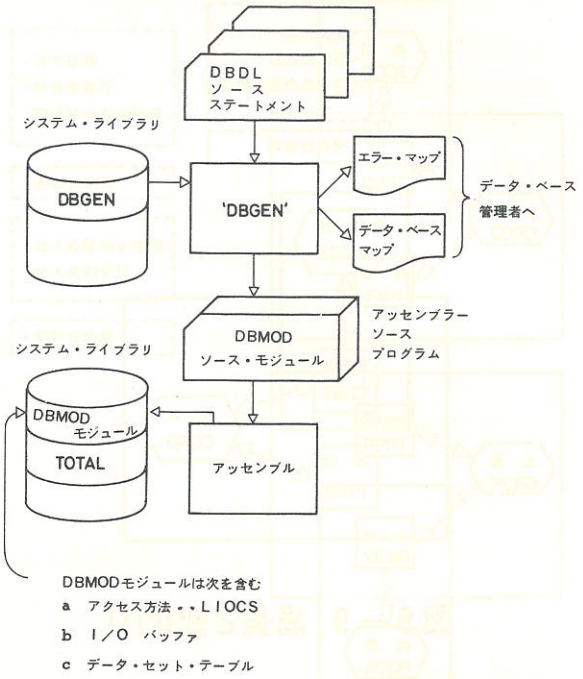


図9－7 データ・ベースの定義手順

最後にデータ・セットが格納されるDASDのエリアをフォーマッティングするために，TOTAL FORMAT プログラムを実行しなければならない。図9－8参照。

以上でデータ・ベースに関する仕事は終り，次はアプリケーション・プログラムを書くことになる。

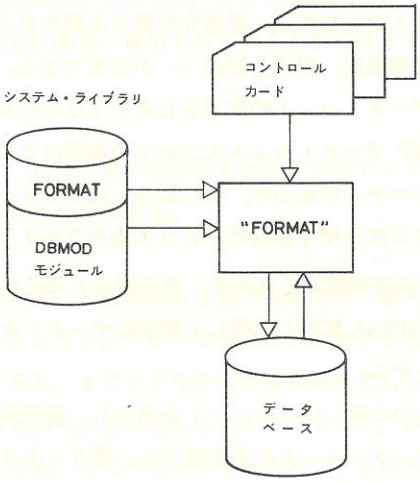


図9－8
データ・ベースのフォーマッティング

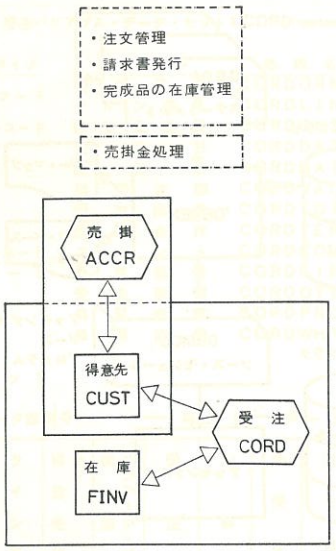


図9－9 成長と変化(1)

成長と変化 (1)

数ケ月かゝっていろいろなアプリケーション・プログラムを完成したと仮定しよう。そうした時に，部長から売掛金勘定処理システムを作れという命令が下された。

ここでデータ処理部門によくある「窮地に直面」したわけである。すなわち現存のシステムに影響が出たのだ。しかし我々はこのシステムをTOTALで作っている。どんな影響が出たか調べてみよう。

まず，各得意先別の売掛金の借方，貸方を記録する新しいファイルが必要である。図9－9参照。しかも売掛金勘定処理システムのためには，得意先に関する更に多くの新しい情報 … 信用限度，売掛金残高 … が必要である。このために得意先データ・セットの変更をしなくてはならない。CORDやFINVデータ・セットについての影響はどうだろうか？当初のシステムのために，既に開発したアプリケーション・プログラムに及ぼす影響は，どうであろうか？

この状況に対応するために，次の事をしなくてはならない。

1. TOTALに対して新しく売掛金データ・セットの定義をする。
2. 4つの新しいエレメントを追加し，得意先マスター・データ・セットを再定義する。図9－10参照。
3. DBMODの再ジェネレーション。
4. 新しく追加された売掛金データ・セット用のDASD

得意先マスター・データ・セット(CUST：customer)

エレメント	処理名	長さ
得意先番号	CUSTCTRL	6
得意先名	CUSTNAME	30
得意先住所	CUSTADDR	30
得意先所在都市	CUSTCTYS	20
得意先郵便番号	CUSTZIPC	5
特別信用条件	CUSTCRED	6
得意先信用限度	CUSTCLIM	7
売掛金残	CUSTARBL	7
売掛(60日)	CUSTAR60	7
売掛(90日)	CUSTAR90	7

フォーマット

ルート 8	得意先番号 6	キイ 受注 8	リンケージ・パス 売掛 ※8	得意先名 30	得意先住所 30	得意先所在都市 20	得意先郵便番号 5	特別信用条件 6	得意先信用限度 ※7	売掛金残 ※7	売掛(60日) ※7	売掛(90日) ※7

↑
※印：追加したフィールド

定義

```
BEGIN-MASTER-DATA-SET:
DATA-SET-NAME=CUST
MASTER-DATA:
CUSTROOT=8
CUSTCTRL=6
CUSTLK01=8=CORD
CUSTLK02=8=ACCR
CUSTNAME=30
CUSTADDR=30
CUSTCTYS=20
CUSTZIPC=5
CUSTCRED=6
CUSTCLIM=7
CUSTARBL=7
CUSTAR60=7
CUSTAR90=7
END-DATA:
```

図9－10 得意先マスター・データ・セット(2)(CUST)

スペースのフォーマティングや得意先データ・セットの変更をする。

5. 得意先データ・セットをダンプし，再びロードし直す。そして売掛金データ・セットをロードする。

これで準備は完了し，売掛金データ・セットを利用するアプリーケーション・プログラムを書くことができる。

当然ながらここの要点は，

1. 既に開発した受注，請求書作成，在庫管理のためのアプリケーション・プログラムにはふれる必要はないし，コンパイルをやり直す必要もない。なぜか？それはこれらのプログラムが要求して受けとるのに必要とするエレメントだけであり，データ・セットのレコード全部ではないから，そして，それらのエレメントには，変更がないからである。CUSTレコードの変更はこれ

らのプログラムに何の影響も与えない。

2. また，得意先データ・セットは影響があり，それを物理的に変更したけれども，得意先データ・セットに直接関連づけられているCORDデータ・セットやFINVデータ・セットは一切の変更を必要としない。再定義および再ロードの必要はない。マスター・データ・セットの変更は他のデータ・セットとは全く独立なのである。

このようにTOTALを使うことによって，我々はモジュールをつくり上げることができるし，現在のアプリケーションプログラムでは予定になった，あるいは予知されなかった変更が生じても，現存プログラムには何ら影響を及ぼすことなくデータ・ベースを発展させることができる。そしてデータ・ベース自身にも，その変更の影響を最小限度にすることができる。

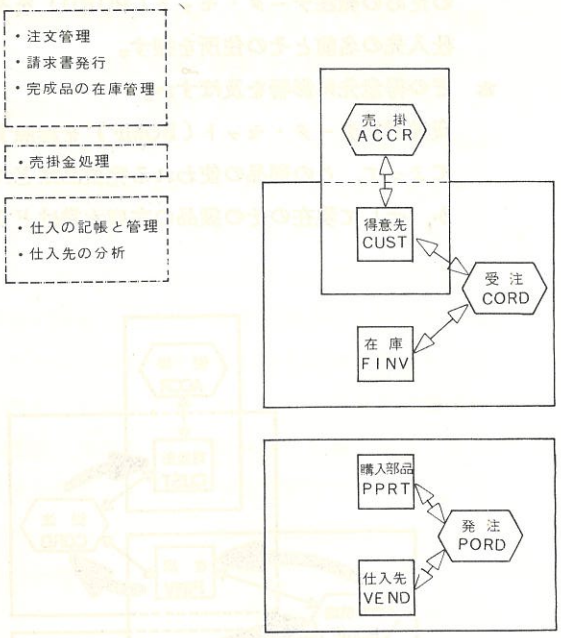


図9－11 成長と変化(2)

成長と変化 (2)

1つのデータ・ベースごとに65,000データ・セットをTOTALは許容できる。そして，データ・ベースの数には制限はない。

おそらく他のグループは仕入の記帳と管理，および仕入先の分析のための仕事をしているであろう。そして，アプリケーション・プログラムは購入部品（PPRT）マスター・データ・セットと仕入先（VEND）マスター・データ・セットを使用し，共に発注（PORD）バリアブル・データ・セットにリンクしているであろう。図9－11参照。

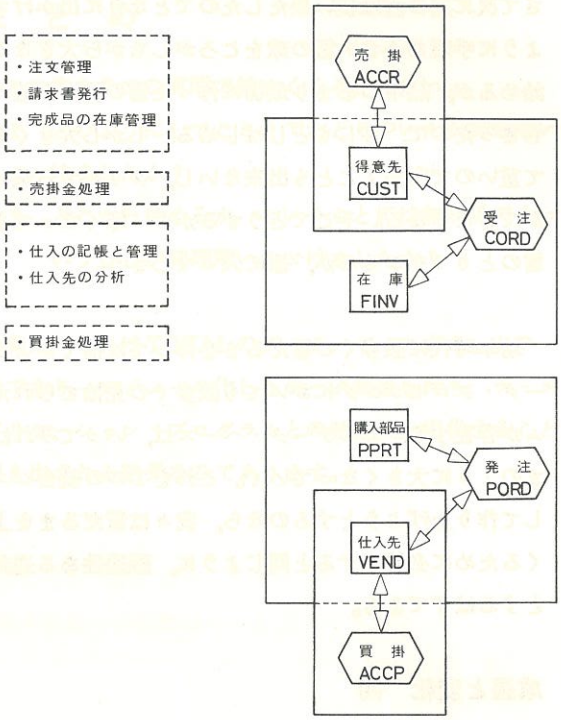


図9－12 成長と変化(3)

成長と変化 (3)

それらが実施されている時に，買掛金勘定処理システムが追加されたらどうなっただろうか。

買掛金勘定処理の必要から，買掛金データ・セットを作り統合したとしよう。図9－12参照。これによるデータ・ベース全体への影響は，前に売掛金データ・セットを追加した場合と同様である。関係のないプログラムやファイルは影響を受けないということが，いくつものデータ・ベースがそれぞれ独立に開発・使用されているときにはますます重要となる。

さて，ここで別々に開発された2つのデータ・ベースができあがったことになる。

子供の頃，初雪が降ると誰でも町内で一番大きな雪だるまを作ろうとする。大きいのを作るには大きな雪の玉をつくる必要がある。まず小さな手頃な雪の球をこしらえてそれをころがしながら，だんだんと大きくしていく。押し切れなくなると「さあ，ここに雪だるまを作ろう」ということになる。

さて次に庭の雪は使い果たしたのでとなりに出かけて，同じように手頃な小さい雪の球をころがしながら大きなのを作り始めるが，熱中のあまり最初に作った雪の塊から離れすぎてしまったので，押しもどしはじめる。しかし大きくなりすぎて重いので動かすことも出来ないし，ソリにすら乗せることができなくなる。そこでどうするか？　寒くなってきたので雪のとりでをつくるか，家に入ってしまう。

幼年時代に数多くの雪だるまを作っては捨てたように，データ・プロセシングにおいても数多くの見捨てられたシステムが存在する。このデータ・ベースは，いってみれば雪だるまのように大きくなっていく。これを１つの統合システムとして作り上げようとするのなら，我々は雪だるまを上手につくるために必要とすると同じように，融通性ある道具を必要とするはずである。

成長と変化　(4)

需要計画と資材原価管理を含む資材管理システムを導入し，先に開発した２つのデータ・ベースとの統合化を行ってみよう。ここで完成品在庫（ＦＩＮＶ）を購入部品（ＰＰＲＴ）と関連させるための資財明細データセット（ＢＯＭＦ）を追加する。図９－１３参照。従って，ＦＩＮＶとＰＰＲＴのＤＢＭＯＤ は再定義されジェネレーションし直して，この２つのデータ・セット（この２つに限りそれ以外はない）をロードし直さなければならない。もちろん，現存のアプリケーション・プログラムには，何ら変更を必要としない。

９つのデータ・セットを統合したデータ・ベースができ上ったわけで，このデータ・ベースを利用してどの様な種類のものができるか考えてみよう。部品番号１２３の部品が安全在庫水準を割ったとしよう。いくつかの質問に答える必要がある。図９－１４参照。

＊　この部品の追加注文をすぐできるか。
　　購入部品データ・セット（ＰＰＲＴ）に部品１２３について直接アクセスして，この部品に関連する発注のための発注データ・セット（ＰＯＲＤ）をみつけて，仕入先の名前とその住所を探す。

＊　どの得意先に影響を及ぼすか。
　　資材明細データ・セット（ＢＯＭＦ）を参照することによって，この部品の使われる完製品はどれであるか，そして現在のその製品の在庫水準はどうかを，

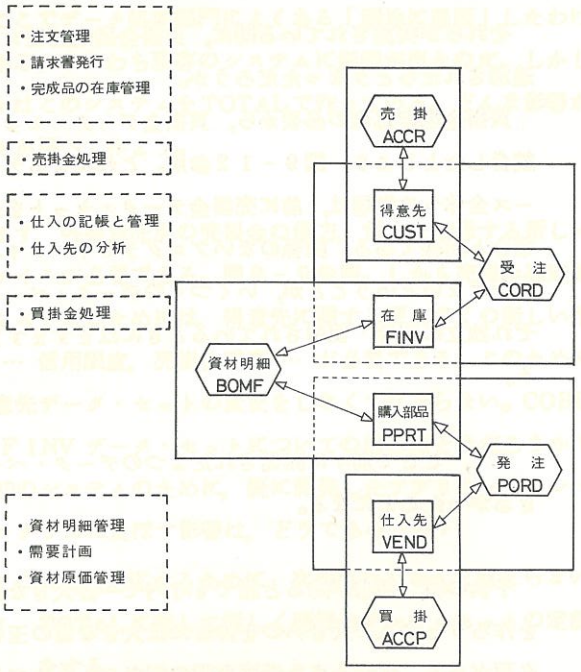


図９－13 成長と変化(4)

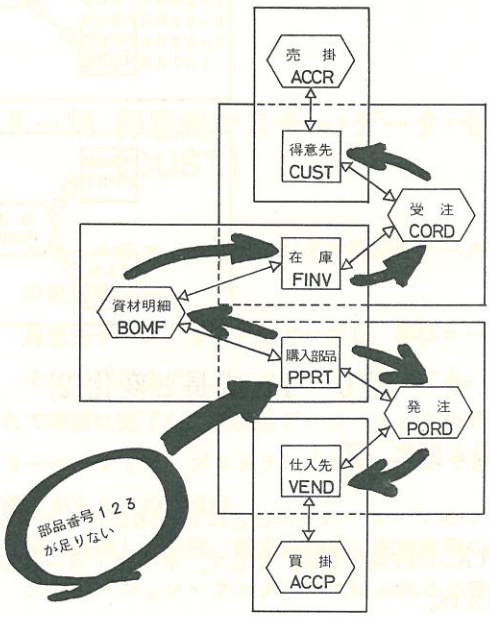


図９－14 データ・ベース使用例

FINV より得て，この製品の受注残を CORD データ・セットで確認することによって，影響を受けそうな得意先を CUST データ・セットから探す。

これがデータ・ベースであり，ＭＩＳである。これこそコンピュータが今日，企業に対してなすべきことではなかろうか。

ま と め

この章を２つの質問で締めくくりとしたい。

1. どのようにしてここまでたどりついたか御理解いただけましたか？
2. どこから始めるか，ということには無関係であったということが御理解いただけましたか？

実は，これはTOTALのデータ・ベースがネットワーク構造であり，モジュール化したデータ構造を持ち，データとアプリケーション・プログラムとの独立性を提供することから引き出される結果なのであります。

第10章　ま　と　め

本書はTOTALデータ・ベースの構造とその管理の方法を紹介したものである。

＊ データ・ベースはマスター・データ・セットと（トランザクション志向の）バリアブル・データ・セットとのネットワークとして構成される。これらのデータ・セット間の関係は，実際上無限といってよいほどの組み合せが可能である。

＊ データ・ベースに対するすべてのコミュニケーションは，ユーザーがホスト言語で書くアプリケーション・プログラムのデータ処理命令とは切り離されている。

TOTALは世界で最も広汎に，かつ成功裡に用いられているデータ・ベース・マネジメント・システムである。それには次の９つの理由が挙げられる。

1. すべてのアプリケーションに利用しうる共通データ・ベースが作られ，維持でき，データ・ベースの変化はアプリケーション・プログラムに変化を与えない。
2. データ・ベースとのコミュニケーション … 読み，書き，追加，削除など … はすべて標準化されており，最高の効率と保全機能，柔軟性を提供している。これはアプリケーション・プログラマーに制限を課することなく達成される。従って，プログラマーは自分のホスト言語アプリケーションの処理ロジックについては全く自由にコントロールできる。
3. すべてのデータ・マネジメントに関するプログラミングとテストは，アプリケーション・プログラミングの側からは不要になる。この作業は統合データ・ベースを自分で作ろうとすれば，恐らく全作業の約８０％を占めるだろう。
4. 拡張可能なデータ・ベース・デザインとシステム・デザイン機能が得られる。例えば，新しいエレメントをレコードに追加したり，データ・セット同志の間に新しい関係を定義したりすることは，現在それらのレコードやデータ・セットを用いているプログラムに何ら影響せずに可能である。古いプログラムを書き直したり，再びコンパイルする必要はない。
5. ハードウエアの面で得られる利益も大きい。２つの例をあげると，
   ＊ Ｉ／Ｏバッファの共有により，コアが節約でき，実効ディスク記憶容量と実行効率を大巾に増大する。
   ＊ 外部ディレクトリーと，インデックスが不要であるから検索および保守の早さが増大し，データ自体に使えるディスク・スペースが増大する。
6. ファイルの重複，および重複データが排除される。
7. TOTALにより機器，オペレーティング・システム，データ・コミュニケーション・モニター，ＤＡＳＤ装置およびホスト言語などの選択に独特の柔軟性が与えられる。年中行事から変換作業が追放される。
8. ユーザーはデータ・セットの構成，データ関連づけや物理的位置などについてコントロールを有しているが，これらに関するプログラミングから解放される。
9. データ・ベースの保護と保全という点からいうと，TOTALのこれらの機能は非常に広汎に渡っており，データ・セットを壊したり，チェインを切ったり，レコードを失ったりまたは現在動いているシステムに悪い影響を与えるような，他の多くのエラーを起したりすることは，事実上起り得ない。さらに，電源が切れたり，ディスクが物理的に壊れたりする場合には，TOTALの広範囲のデータ・ベース・ロギングや再スタート機能を使って，データ・ベースを数分の内に復旧し再スタートできる。

図９－14　データ・ベース使用例

株式会社 アシスト

東京都港区西新橋３－４－１　エム・ワイ・ビル　〒105　電話（03）434－5986