56－S 002

マン・マシン・ユーザ・インタフェイス
に関する調査研究報告書

昭 和 57 年 3 月

JIPDEC

財団法人　日本情報処理開発協会

この報告書は、日本自転車振興会から競輪収益の一部である機械工業振興資金の補助を受けて昭和５６年度に実施した「マン・マシン・ユーザ・インタフェイスの調査研究」の成果をとりまとめたものであります。

はじめに

当財団では情報処理技術の調査研究の一環として、昭和５５年度より「マン・マシン・ユーザ・インタフェイスに関する調査研究」に着手した。

今後の情報処理に要求される基本的条件の１つは、より広汎な分野およびアプリケーションに対応し得るマン・マシン・ユーザ・インタフェイスの改善である。即ち、従来のコンピュータ・システムの使用者は比較的専門家の比率が高く、コンピュータが機械であるための使用上の不自由さや不便さは、ある程度見過すことも可能であった。しかしながら、今後期待される多様な且つ広汎な利用分野においては恐らく数多くの非専門家が直接システムの操作を行う可能性が極めて高いものと考えられる。

情報処理システムが多くの非専門家にとって使い易いツールとなるには、コンピュータ側のインテリジェンスを向上させ、人間の五感による情報伝達機能により近い形態で各種の情報の処理が容易に行なえることが必要不可欠となる。

そこで、本プロジェクトにおいては、１９８５年頃を目途とした今後の新しい情報処理システムにおいて、実現されねばならぬ、マン・マシン・ユーザ・インタフェイスの革新的な向上を支える基本的諸技術についての調査研究を行うこととした。

本調査研究は、昭和５５年度より３ケ年計画で実施する予定であり、本報告書はその第２年度目の成果、即ち、マン・マシン・ユーザ・インタフェイス・システムにおいて中核となる分散型データベースシステムの実現と応用についての技術的検討と一部のモデル化についてまとめたものである。

本報告書がこの方面に興味をお持ちの方々に広く利用され、今後の情報処理技術向上の一助として寄与することができれば幸いである。

昭和５７年３月

目　　次

1．概　　論

オフィス情報システム（ＯＩＳ）をはじめとする新たな情報システムは、ユーザに対して、容易なマン・マシン・ユーザインタフェースとして存在する必要がある。これらのシステムに於いて、種々の通信ネットワークを利用した分散処理が、データベースシステム利用を中心に行われていくものと考えられている。

この背景には、ＶＬＳＩを中心とした計算機ハードウェア技術の進歩、広域パケット交換ネットワークとローカルネットワークによる通信技術の進歩、そして、高度なデータ独立性を達成しつつあるデータベースシステム（ＤＢＳ）技術の進歩とを挙げることができる。ＶＬＳＩの進歩によって、従来の大型、超大型機で行われていた、計算処理のほとんどは、各ユーザがこれらの計算機と同等な能力を備えたパーソナル計算機システムを専有する事によって、なされてしまうと考えられている。こうした専有化の中で、複数ユーザによって共有される最後のリソースは、データベースである。この意味で、新たな情報システムに於いて、複数ユーザ下でのデータベースの総合的な管理技術が、中核の技術となるものである。

新情報システムは、複数の計算機システム（e.g.パーソナル計算機システム、特殊専用計算機システム）が、通信ネットワークで相互に通信可能な様に結合された形態をとる。このシステムは、ある組織（e.g.会社の部、課）毎のローカルな結合と、組織間のグローバルな結合とから成る。前者のローカルな結合は、主にこの組織内での利用が中心となる。後者では、組織間での通信と相互の情報利用が行われる。ローカルな結合は、Ethernet〔ＮＥＴＣＲ７６〕の様なローカルネットワーク（local（area）network）によって各種の計算機システムが相互結合される。グローバル結合では、ＡＲＰＡＮＥＴ、ＤＤＸといった広域のパケット交換ネットワークによって、各ローカルなネットワークシステムが結合されることになる。この様に、新情報システムは、従来の広域パケット交換ネットワークに加えて、新たなローカルネットワークの利用を考慮する必要がある。

新情報システムに於いて、データベースが主要なリソースとなると先に述べた。組織体が用いるデータは、従来の集中システムの様に単一のシステム内で、全て集中管理する替りに、相互結合された複数の計算機システムで分散して保持する可能性がある。ＯＩＳの様な、新たな情報システムに於いては、データは従来の様に、強力に集中して管理されるよりも、各ローカルな利用サイトでの利用が高まり、これらのデータ上に、ゆるい全体的な管理機能が存在することが望ましい。これによって各利用サイドでの、データの利用効率が高まる。従って、データは、各計算機システム内での物理的な冗長保持も考慮した、種々の分散形態を取ることになる。この中には、各サイトで自立的に生成、管理されてきたデータを、システム全体に組み込むことも必要になる。

従来のデータベース利用は、検索業務を中心とした静的な利用形態が取られていた。これに対して、今後のオフィス情報システム（ＯＩＳ）を中心とした新情報システムでは、データの更新、スキーマの動的な再定義を必要とする動的なデータベース利用が中心となっていくと考えられる。分散型デー

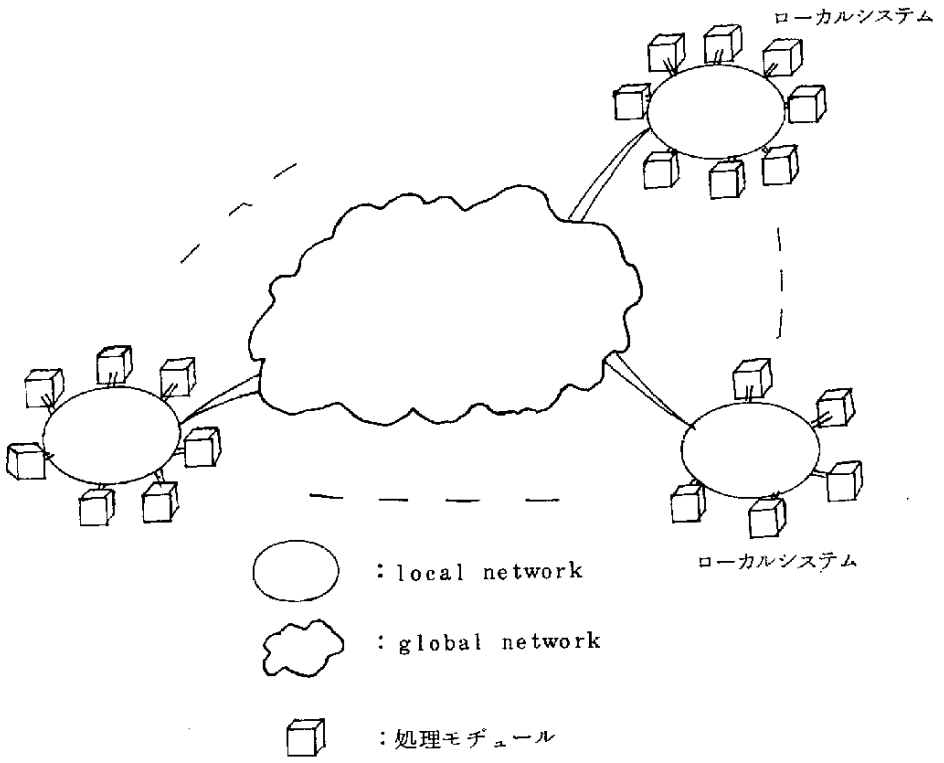


Fig 1.1 情報システム

タベースシステム（ＤＤＢＳ）における更新演算の処理は、次の２つのステップによってなされる。

(1) 更新の分散処理

(2) 各データベースシステム（ＤＢＳ）サイトでのローカルな更新処理

(1)では、複数のＤＢＳサイトに対する更新演算は、ユーザレベルでの原子的な処理単位(transaction)の原子性（ａｔｏｍｉｃｉｔｙ）を保障し、かつ各ＤＢＳ内、及びＤＢＳ間のインテグリティ条件を保持する様になされねばならない。この更新を複数ユーザ下で、行なわさせる為の（分散）同時更新制御（ｃｏｎｃｕｒｒｅｎｃｙ ｃｏｎｔｒｏｌ）〔ＢＥＲＮＰ７８、８１ａ、８１ｂ〕と、ＤＤＢＳの種々の障害（各ＤＢＳの障害と、通信ネットワークの障害）とに対しても種々のインテグリティ条件を保つ為の障害回復（ｒｅｃｏｖｅｒｙ）管理とが必要になる。

各ＤＢＳサイトでは、ＤＤＢＳ全体で共通形式の更新演算を、そのＤＢＳで実行可能な形式に変換し、実行させる必要がある。この為には、各ＤＢＳサイトで異なったデータモデル間でのスキーマの等価変換と、演算の変換と実行を行う必要がある。更新演算の実行では、分散更新処理と同時に、複数ユーザ下での同時実行制御と、障害に対する管理機能が必要である。又、分散更新処理に於ける更新の単位（e.g. lockの単位）と、各ＤＢＳに於ける更新単位との整合性も求められる。

以上より、新情報システムに於いて、以下の点を特徴とする分散型データベースシステム（ｄｉｓｔｒｉｂｕｔｅｄ ｄａｔａｂａｓｅ ｓｙｓｔｅｍ ｏｒ ＤＤＢＳ）が、その最も中核の技術となるものである。

(i) 種々のネットワーク、特にローカルネットワークによる通信

(ii) 個人用のデータベースシステム

(iii) 異種データベースシステム

(iv) データベースの動的利用

本プロジェクトでは、上記の特徴を有した新たな分散型データベースシステム、即ち第Ⅱ期Jipdec分散型データベースシステム（ＪＤＤＢＳ－Ⅱ）〔ＴＡＫＩＭ８０、８１ａ、ｂ、ｃ〕の実現に向けた技術検討を３カ年にわたって行っている。

本プロジェクトの初年度に於いては、ユーザの利用面からみた将来の情報システムとのインタフェース技術の動向と問題点の整理を行いＪＤＤＢＳ－Ⅱの全体的アーキテクチャを確立し、その一部のモデル化を行った。

本年度は、２年度目にあたり、以下の調査研究を行った。

(1) 異種ＤＢＳ、即ちＣＯＤＡＳＹＬデータベースシステムに対する高水準非手続的な検索利用リレーショナルインタフエースシステムＬＤＰ－Ｖ1.5（昨年度実現）の性能、機能、及びユーザの利用面からの総合評価を行い、この有用性を示した。

(2) (1)は検索利用インタフェースであるが、更に更新機能を備えたインタフェースシステム（ＬＤＰ－Ｖ２）の実現を行った。

(3) ローカルネットワークの１：Ｎ通信機能を用いた、有効な分散問合せ処理方式を考案した。

(4) ＤＤＢＳ及びＤＢＳに於けるデータの更新技術の整理を行った。

第２章では、ＪＤＤＢＳの全体構成と問合せ及び更新演算の分散処理方式と、その応用について論じる。

第３章では、ＬＤＰ－Ｖ1.5の設計、実現、評価について述べる。

第４章では、ＬＤＰ－Ｖ２の設計と実現について論じる。

２．分散型データベースシステム

本章では、複数のデータベースシステムを通信ネットワークを介して統合する分散型データベースシステム（ＤＤＢＳ）について論じる。ＤＤＢＳは、ユーザにとって通信ネットワークで結合されたＤＢＳの集合としてではなく１つの仮想的なデータベースシステム（１つのスキーマと言語）として、サポートされねばならない。従って、ＤＤＢＳを実現する為には、複数のデータベースシステム（ＤＢＳ）が通信ネットワークによって結合されることによって生ずるＤＢＳの分散性の問題と、各DBSが種々のデータモデル（データ構造とアクセス言語）によってサポートされる事によるＤＢＳの異種性の問題とが解決されねばならない〔ＴＡＫＩＭ７８、７９〕、この２つの問題は、ＪＤＤＢＳ－Ｉ〔ＪＤＤＢＳ８０〕における四層スキーマ構造（four－schema structure）〔ＴＡＫＩＭ７８、７９〕に基づいたスキーマ階層によって解決できる。

最近の情報システムに大きな影響を与えている技術としては、以下の２つがある。

(i)　ローカルネットワーク

(ii)　パーソナル計算機システム

オフィス情報システム（ＯＩＳ）等の新たな情報システムは、これらを主要な構成要素として構成され様としている。(1)は、ＤＢＳ間の通信として、物理的な１：N通信機能をサポートしている点が、ＤＤＢＳに、大きなインパクトを与えている。

データベースシステム（ＤＢＳ）に於ける利用動向としては、次の２点がある。

(i)　リレーショナルＤＢＭＳの商用化が進み、リレーショナル言語による非手続的検索利用が普及。

(ii)　更新要求の増大

(ii)では、複数ユーザ下でデータベースのインテグリティ、セキュリティの保持が必要となる。DDBSでは、更に、異なったサイト間でのデータのインテグリティの保持を、通信ネットワークを用いて行うことが求められる。

我々の目指す分散型データベースシステム（ＤＤＢＳ）は、ＪＤＤＢＳ－Ⅱと呼ばれ、以下の特徴を有している。

(i)　ローカル放送ネットワーク（e.g.Ethernet）の利用

(ii)　種々のデータベースシステム（e.g.個人用ＤＢＳ、既存大型ＤＢＳ）の組み込み

(iii)　動的なデータ更新のサポート

本章では、ＪＤＤＢＳ－Ⅱの全体アーキテクチャ（2.1）、分散問合せ処理（2.2）、分散更新処理（2.3）、ＪＤＤＢＳ－Ⅱの応用（2.4）、について述べる。

2.1　ＪＤＤＢＳ－Ⅱの全体アーキテクチャ

本節では、第２期分散型データベースシステム、ＪＤＤＢＳ－Ⅱの全体アーキテクチャについて述べる。

2.1.1 スキーマ構造

ＤＤＢＳでは、分散と異種の問題を解決する為に、図2.1に示す四層スキーマ構造（four-schema structure）〔ＴＡＫＩＭ７８，７９〕が必要となる。四層スキーマ構造は、次の４つのスキーマ層から成る。

(i) ローカル内部スキーマ（ＬＩＳ）

(ii) ローカル概念スキーマ（ＬＣＳ）

(iii) 全体概念スキーマ（ＧＣＳ）

(iv) 外部スキーマ（ＥＸＳ）

ローカル内部スキーマ（ＬＩＳ）は、ＤＤＢＳを構成する各ＤＢＳが、ＤＤＢＳに於いて利用可能なデータの記述である。各ＤＢＳに固有なデータモデルに基づいている。ＤＢＳの異種性とは、データモデル（i. e. データ構造とアクセス言語）の相違と定義する。ローカル概念スキーマ（ＬＣＳ）とは、LISを、ＤＤＢＳ全体で共通なデータモデルによって記述したものである。このレベルに於いて、ＤＢＳの異種性は解決された事になる。

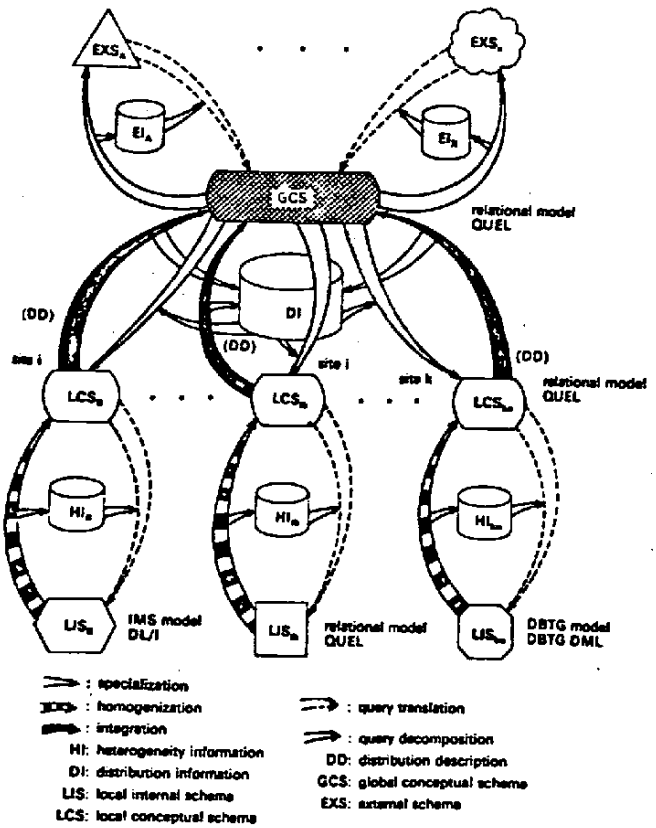


図2.1 Four-Schema Structure(FSS)

全体概念スキーマ（ＧＣＳ）は、各ＬＣＳから、ＤＤＢＳに対して一意な意味のあるデータを記述したものである。ＤＤＢＳは、このレベルに於いて、１つの論理ＤＢＳに仮想化された事になる。即ち、ユーザは、各ＤＢＳの相違、その所在を意識する事なく、あたかも１つのＤＢＳの様にアクセス出来る。外部スキーマ（ＥＸＳ）は、ＧＣＳに対して、各アプリケーションの必要とするデータを適したデータモデルで表したものである。

各スキーマ層の記述及びスキーマ間の対応情報は、３種のＤＤ／Ｄ、i. e. 異種性情報（ＨＩ）、分散情報（ＤＩ）、外部情報（ＥＩ）内に格納される。ＨＩは、ＬＣＳ、ＬＩＳの記述と共に、この間の対応情報が格納される。ＤＩは、ＧＣＳと各ＬＣＳの対応情報と、ＧＣＳの記述とが格

納される。ＥＩは、ＥＸＳの記述と、ＥＸＳとＧＣＳとの対応情報が格納される。

ＧＣＳは、異種及び分散の問題が解決されたスキーマであるので、ＤＤＢＳの基本機能はGCSにおいて実現されたことになる。この為、ＬＩＳ、ＬＣＳ、ＧＣＳについて考えることにする。又、ＤＤＢＳの共通モデルとしては、関係モデル〔ＣＯＤＤＥ７０〕を用いる事にする。アクセス言語としては、関係計算言語ＱＵＥＬ〔ＨＥＬＤＧ７５〕を考え、検索アクセスについてだけ考える。

2.1.2 システムアーキテクチャ

分散型データベースシステム（ＤＤＢＳ）は、通信ネットワークで結合されたデータモヂュール（ＤＭ）から成っている。各データモヂュール（ＤＭ）は、次の５つのサブモヂュールから成っている〔図2.2〕。

(1) ＤＢＳ（database system）

(2) ＧＤＰ（全体ＤＢプロセッサ）

(3) ＬＤＰ（ローカルＤＢプロセッサ）

(4) ＲＷＳ（リレーショナル作業領域）

(5) ＤＤ／Ｄ

ＤＢＳは、データベース管理システム（ＤＢＭＳ）と、データベース（ＤＢ）から成る。ＤＢＳは、ＤＤＢＳの最も基本となるデータベースアクセス機能を提供している。ＤＢＳとしては、以下のものを考える。

(i) 既存データベースシステム（e.g. ＣＯＤＡＳＹＬ型のＤＢＳ）

(ii) 個人用の（小型）データベースシステム

ＬＤＰは、ＤＢＳのデータモデルが、共通データモデル、i.e. リレーショナルモデル、でない場合に、共通－ローカルの変換器の役目を持つ。この時、変換情報は、ＤＤ／Ｄの異種性情報（ＨＩ）を用いる。ＬＤＰで得られた結果は、ＲＷＳに、リレーションとして格納される。

ＧＤＰは、ＤＤＢＳ内の各データモヂュール（ＤＭ）にまたがった問合せを処理する為に必要となる通信処理の制御を行う。ＧＤＰの必要とする情報は、ＤＤ／Ｄの分散情報（ＤＩ）である。分散情報は、各データの所在、及びパフォーマンス情報から成っている。通信処理の為のデータは、リレーショナル作業領域にリレーションとして格納される。

ＤＤＢＳ全体のシステム構成を図2.2に示す。図中で、箱は各処理モヂュールを表し、三角形は、ＤＤ／Ｄを示している。

ＤＤＢＳに於いて、通信ネットワークは、各データモヂュール（ＤＭ）間で、データ及び制御情報を転送する為に用いられるので重要である。通信ネットワークとしては、次の２種がある。

(i) 広域パケット交換ネットワーク

(ii) ローカルネットワーク

広域パケット交換ネットワーク（e.g. ＡＲＰＡＮＥＴ）は、１対の通信実体間の通信手段を提供することが特徴である。

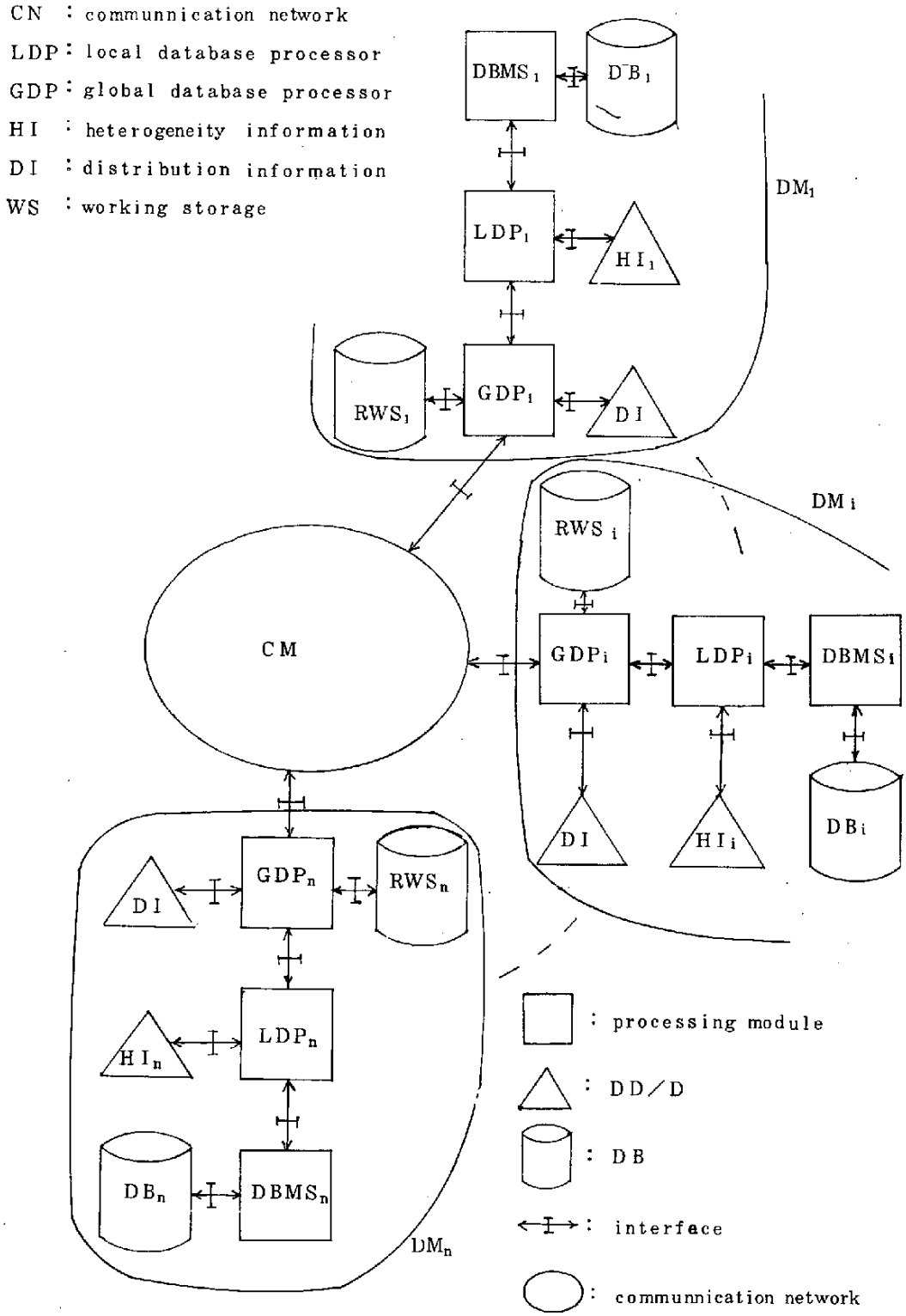


図2.2　JDDBS－Ⅱの概要

これに対して、ローカルネットワーク（e.g. Ethernet）は、1つの通信実体と、N個の通信実体間の1対N通信機能を提供できる。他に前者に対して、高速性（～1MBps）、高信頼性、近距離性の特徴を有している。

2.2 分散問合せ処理

分散型データベースシステム（DDBS）において、各データベースシステム（DBS）の異種性と分散性と独立な全体概念スキーマ（GCS）内のリレーション（GCS relation）に対するQUEL問合せ（GCS問合せと呼ぶ）は、各データモデュールと、これらを結合する通信ネットワークによって通信ネットワークによって処理され、結果がユーザに出力される。

この為には、以下の3種の処理が必要となる。

(1) 表現変換

(2) DM間通信処理

(3) DM内ローカル処理

(1)では、GCSリレーションを参照するGCS問合せから、対応するLCSリレーションを参照する問合せ（これを全体LCS問合せ（global LCS query）と呼ぶ）が生成される。即ち必要なデータの存在する所在を意識した問合せが生成されることになる。

(2)では、複数のデータモデュール（DM）にまたがった問合せ（e. g. DM間での結合）を処理する為に、通信ネットワークによって結合されたDM間での通信と処理が行われる。ここでは、通信コスト、応答時間等を最少とする様に、DM間での通信処理が行われる。

(3)では、各DM内に格納されたデータに対するローカルな処理がなされる。DMのDBSが、リレーショナルDBSでない場合には、QUEL問合せからこのDBSで実行可能な形式への変換と、その実行が必要になる。

2.21 表現変換

GCS問合せは、QUELによって記述されたGCSリレーションに対する問合せである。GCSリレーションは、LCSリレーション上のビュー（view）リレーションと考えられる。ビュー・リレーションは、LCSリレーションに対する制限、射影、結合によって定義されるものとして、和（union）、差（difference）、積（intersection）については考えないことにする。

GCS問合せは、GCSリレーションの定義式を用いて問合せ修正（query modification）〔STONM76〕手法によって、全体LCS問合せに変換出来る〔図2.3〕

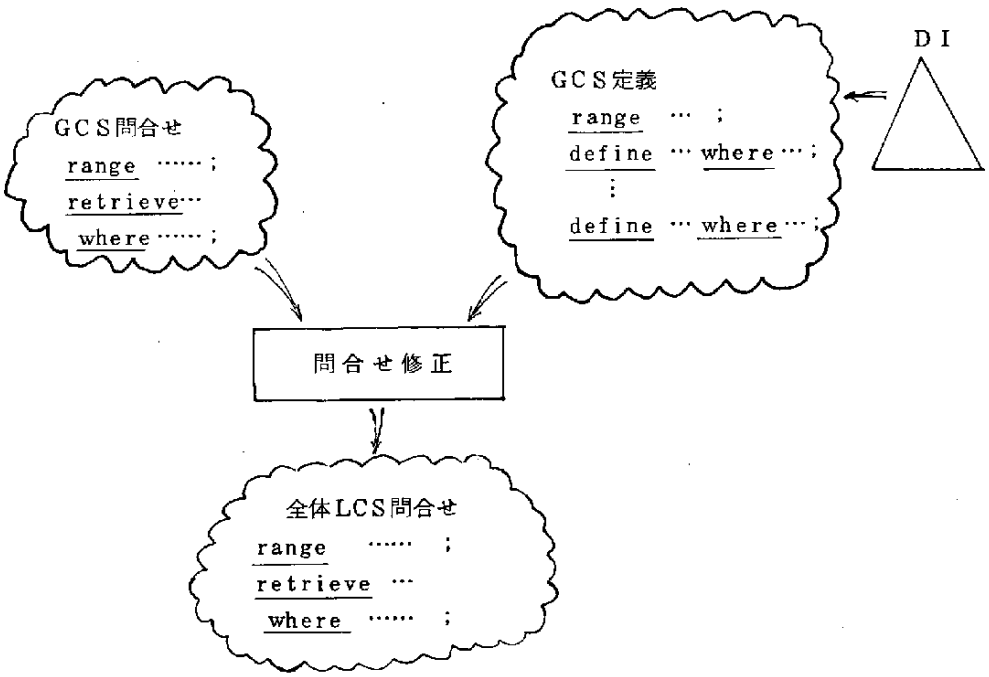


図2.3　表現変換の概要

A．GCSリレーションの定義

GCSリレーションは、各DM上のLCSリレーション上のビューリレーションとして定義される。ビューリレーションは、以下の様にして定義される。

$$\left.\begin{array}{l} \mathrm{drange}(l_1, L_1 : d_1) \cdots\cdots (l_m, L_m : d_m); \\ \mathrm{define}\ G\ \langle G_1\text{-def}\rangle : \\ \qquad\qquad \langle G_2\text{-def}\rangle : \\ \qquad\qquad\quad \vdots \\ \qquad\qquad \langle G_n\text{-def}\rangle ; \end{array}\right\} \quad (1)$$

$L_1, \cdots, L_m$は、各々$DM_{d_1}, \cdots, DM_{d_m}$に存在するLCSリレーションである。各$L_i$は、互いに必ずしも異なっている必要はない。

l_iは、LCSリレーションL_iに対する組変数である。この様にdrange文は、LCSリレーションと、GCSリレーション定義内で用いられる組変数との対応を示している。

Gは、定義する全体概念スキーマ（GCS）の名前である。各<Gj－def>は、次の様なGCSリレーションGjを定義している。

$$\left.\begin{array}{l} \langle Gj\text{-def}\rangle :: = Gj(ga\text{-}def_{j_1}, \cdots, ga\text{-}def_{jk_j}) \\ \qquad\qquad \mathrm{where}\ gual_j ; \\ ga\text{-}def_{j1} :: = ga_{j1}\,[/GA_{j1}]\,[/format_{j1}] = gaexp_{j1} \end{array}\right\} \quad (2)$$

Ｇｊリレーションのスキームは、$Gj(ga_{j_1}, \cdots, ga_{jk_j})$である（$j=1, 2, \cdots, n$）。ＧＣＳリレーションＧｊの属性$ga_{j_1}$は定義域$GA_{j1}$、形式$format_{j1}$（タイプとバイト長）を持つ。

ＧＣＳ定義式内で、変数は、１．ａという形式で表わされる。１はＬＣＳリレーションＬに対する組で、ａはＬの属性である。１．ａは値として、組変数１が取る組１のａの値，i.e.１〔a〕を持つ。$gaexp_{j1}$は、ＧＣＳ属性ga_{j1}の値を与える機能を持ち、条件式$gual_j$を満足する組変数$l_1, \cdots, l_m$から成る変数（$l_i.a_i$）上の算術式である。$gual_j$は、これらの変数上に述語論理形式に定義された条件式である。

ここで、以下の仮定をＧＣＳ定義に対して設ける。

(i) aggregate関数（e.g. count, max）を含まない。

(ii) $gual_j$は、次の様に積正規化されている。

$$gual_j ::= C_{j1} \wedge C_{j2} \wedge \cdots \wedge C_{jl_j}$$

$$C_{ji} ::= pred_{ji1} \vee \cdots \vee pred_{jihji}$$

ここで各C_{ji}内の全ての述語、$pred_{jik}$（$k=1, 2, \cdots, h_{ji}$）は、同一の組変数を参照している。

例とし、図2.9について考えよう。図は、従業員の情報を表すＥＭＰリレーションがDM_1に、主題と、プロジェクトとレポートの関係性を示すＳＵＢＪリレーションと、各プロジェクトの製品を表すＰＲＯＤリレーションがDM_2に存在することを示している。更に、種々のレポート情報を保有するＲＥＰリレーションがDM_3に存在し、DM_4は、プロジェクトとそのメンバの関係性を示している。これらのＬＣＳリレーションに対して、全体概念スキーマ（ＧＣＳ）ＥＮＴＥＲＰＲＩＳＥは、図2.5の様に定義できる。図2.4に於いて、@のついた属性は、主属性〔ch. 3〕である。

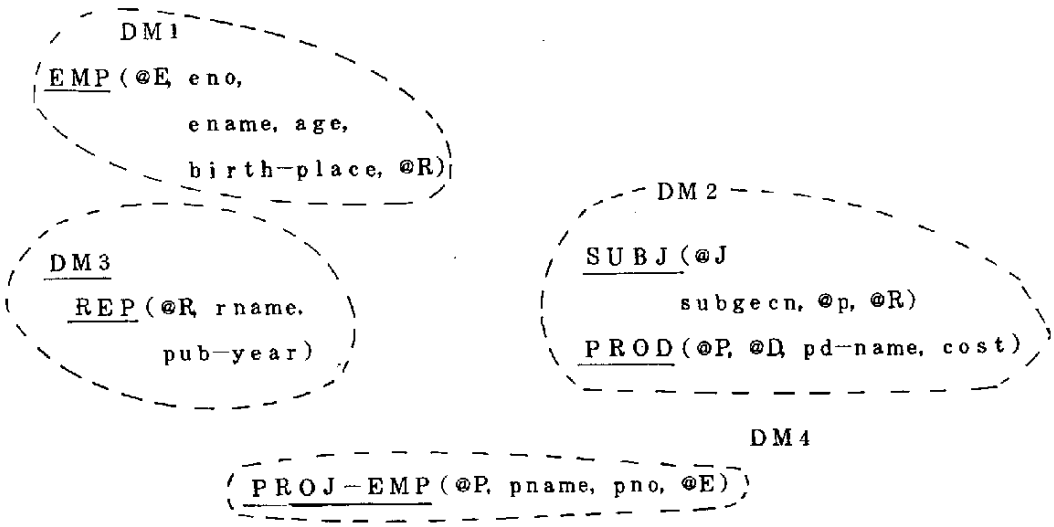


図24　ＬＣＳリレーション

```
range(e, EMP：1)；
range(j, SUBJ：2)(d, PROD：2)；
range(r, REP：3)(pe, PROJ－EMP：4)；
define ENTERPRISE
    EMP－REP(e. @E, r. @R, e. ename, r. rname, r. pub－year)
      where e. @R=r. @R:
    PROJ  (pe. @p, e. @E,  j. @J,
           pe. pname, e. ename, d. pd－name, d. cost, j. subjectn)
      where
           pe. @E=e. @E and pe. @p=d. @p and
           j. @p=pe. @p:
    REP－SUBJ(r. @R, j. @J,
              r. rno, j. subjectn)
      where
           r. @R=j. @R;
```

図2.5　GCSの定義

B．GCS問合せ

GCS問合せは、GCSリレーション上のQUEL問合せ〔ch. 3〕であり、次の様に記述される。

```
range(g1, G1)…(gm, Gm)；
retrieve into R(a1=aexp1, …; ak=aexpk)        }……(1)
where     qual；
```

ここで、G_1 …, G_mは、m個のGCSリレーションである。これらは、互いに異なっている必要はない。g_iは、GCSリレーションG_iに対する組変数である。

問合せは、m個の変数g_i, …, g_mを参照している。Rは、結果リレーション名で、そのスキームは$R(a_1, \cdots; a_k)$である。$aexp_j$は、組変数によって修飾された属性、g. a（gは組変数、aはGの属性）上の算術式である。qualは、問合せの条件式を述語論理形式に表したものである。

分散問合せ処理を考える上で、以下の制約を、QUEL上に設ける。

(1) aggregate関数（e. g. sum, count, max）を、含まない。

(2) 条件式qualは、次の様に積正規化されている。

$$qual ::= C_1 \wedge C_2 \wedge \cdots \wedge C_n$$

$$c_i ::= pred_{i1} \vee pred_{i2} \vee \cdots \vee pred_{il_i}$$

各c_i内の全ての$pred_{ij}$は、皆同一の組変数を参照しなければならない。

図2.5のGCSに対するGCS問合の例として、"分散型データベースを研究しているプロジェクトのメンバの書いた論文を求めよ"は、図2.6様にQUELによって表せる。

```
range(er, EMP－REP)(p, PROJ);
retrieve into RSLT(er.ename, p.pname)
 where
      er.@E＝p.@E  and
      p.subjectn＝"DISTRIBUTED  DATABASE
                                SYSTEMS";
```

図2.6　GCS問合せ

C．問合せ修正

GCS問合せは、GCSリレーション定義式を用いて、問合せ修正〔STONM76〕(query modification)手法によって、対応するLCSリレーションを参照する全体LCS問合せに変形できる。GCSリレーションは、(1)の様に定義されているとする。

```
range(l1, L1 : S1)(l2, L2 : S2)…(lm, Lm : Sm);
define G
    G1 (ga1 1=gexp1 1 ····; ga1k1 = gexp1 l1 )where  ual:
    ⋮
    Gn (gan1 =gexpn1 , …; gank n=gexpnln)where qualn;
```

…(1)

これに対して、(2)の様なGCS問合せが出されたとする。

```
range(g1, G1)(g2, G2)·····(gn, Gn);
retrieve into R(a, =aexp1, ····; ak=aexpk)
    where qual;
```

……(2)

ここで、文字列をEとする時置換$\{\alpha_1/\beta_1, \cdots\alpha_p/\beta_p\}$Eを、E内の変数、$\beta_1 \cdots; \beta_p$を同時に、式$\alpha_1, \cdots, \alpha_p$で各々置換したものとする。

問合せ変形は、次の様に表せる。

```
range   ∪          ∪          βl
      ∀g∈{g1, …; gn} ∀l∈var(G)
retrieve into R (a1 =σ aexp1, …; ak= σaexpk)
    where  σ  qual
           n
           ∧
          i=1   βi quali;
```

……(3)

ここで、

$$\sigma = \{\ \beta_1\ \ gexp_{11}/g_1 \cdot gexp_{11},\ \cdots,\ \beta_1\ \ gexp_{1l_1}/g_1 \cdot gexp_{1l_1},$$

$$\vdots$$

$$\beta_n\ \ gexp_{n_1}/g_n \cdot gexp_{n1},\ \cdots;\ \beta_n\ \ gexp_{nl_n}/g\ \cdot gexp_{nl_n}\ \}$$

ここで、ＧＣＳリレーション、G_iの定義式内で参照しているＬＣＳリレーションの変数集合を、$var(G_i)$とする。(3)式の中で、異った変数g_iとg_jとに対して、各々対応するrangeＧＣＳリレーションG_iとG_jとが同一であるとする。この時、$var(G_i)$内のＬＣＳ変数と、$var(G_j)$内のＬＣＳ変数とは、異った意味を表さねばならない。この為、もし$G_i = G_j$ならば、

$$\beta i = \{ \quad \bigcup_{\forall l \in var(G_j)} \{\beta i l / l\} \ \}$$

$$\beta i = \{ \quad \bigcup_{\forall l \in var(Gj)} \{\beta j l / l\} \ \}$$

である。もし$Gi = Gj$たるGjが存在しなければ、

$$\beta i = \{ \quad \bigcup_{\forall l \in var(G_i)} \{ l / l\} \ \}$$

である。

例えば、以下のＧＣＳ定義があったとする。

```
drange (l, L:1)(m, M:2)(n, N:3):
define G
        A(a1 =l. a, a2 =m. b)
         where
                l. c=m. c ;
        B(b1 =n. d)
         where
                n. e≧2500:
        C(c1 =l. a, c3 =n. e)
         where
                l. f=n. d ;
```

これに対して、次のＧＣＳ問合せが出されたとする。

```
range(a, A)(b, B)(c, C)(d, A);
retrieve into R(a. a1, d. a2)
 where
        a. a1 =b. b1  and a. a2 =c. c2  and
        c. c1 =d. a1  ;
```

ここで、同一ＧＣＳリレーションＡに対して、２つの異なった変数ａとｄとが定義されている。

Aは、ＬＣＳ変数１とｍとを参照している。Ａが２つの異なった使われ方をしている事を明らかにする為には、問合せ修正される時のＬＣＳ変数を各々で区別しなければならない。

この為、$\beta a = \{ a1/1, am/m \}$, $\beta d = \{ d1/1, dm/m \}$ とする。

同様に、置換σは次の様になる。

$$\sigma = \{ \beta a \quad 1.a/a.a_1, \beta a \quad m.b/a.a_2,$$
$$\beta d \quad 1.a/d.a_1, \beta d \quad m.b/d.a_2,$$
$$n.d/b.b_1, \quad 1.a/c.c_1, \quad n.e/c.c_2 \}$$
$$= \{ a1.a/a.a_1, \quad am.b/a.a_2$$
$$d1.a/d.a_1, \quad dm.b/d.a_2$$
$$n.d/b.b_1, \quad 1.a/c.c_1, \quad n.e/c.c_2 \}$$

この結果、ＧＣＳ問合せから、次の様な全体ＬＣＳ問合せが生成される。

```
range(a1, L:1)(d1, L:1)(1, L:1)
range(m, M:2)(n, N:3);
range(am, M:2)(dm, M:2);
retrieve into R(a1.a, dm.b)
  where
        a1.a1=n.d and am.b=n.e and
        1.a=d1.a and
        a1.c=am.c and d1.c=dm.c and
        n.e≧2500 and 1.f=n.d;
```

ここで、更に問題が起こる。上例から解かる様に、ＧＣＳリレーションＣで参照しているＬＣＳ変数１と、ＧＣＳリレーションＡで参照しているＬＣＳ変数１との関係である。上例では、Ａに対する１は、全体ＬＣＳ問合せでは、ａ１とｄ１と名前が変えられ、異った組変数となっており、この１はそのままである。即ち、ａ１≠ｄ１≠１。他の可能性としては、ⅰ）ａ１＝１，ｄ１≠１，ⅱ）ａ１≠１，ｄ１＝１があり得る。各々は、異った問合せの意味を表すことになる。これに対する１つの解は、１つの問合せ内で、どのＧＣＳリレーションに対しても、複数の組変数の定義を禁止する事である。この時の解は、(3)式でβiが不要となり、実現は容易である。

次の解は、ＧＣＳリレーションＧｉとＧｊとが、定義式内で同一ＬＣＳ変数１を参照している時、ユーザにこの変数の共有を意識させることである。この方法は、ＧＣＳの背後に存在するＬＣＳをユーザに意識させることになり、利用の容易性がそこなわれてしまう。

他の解は、各ＧＣＳリレーションの参照するＬＣＳリレーションの組変数は、ＧＣＳリレーション毎に独立と考える方法である。即ち、先の例のＧＣＳリレーションの定義式内で、GCSリレーションＡの参照するＬＣＳ変数１と、Ｃの１とは全く別のものと考えることである。この方法も、実現は、最初の解と同様に容易である。

問題は、ＧＣＳの定義に於いて、ＬＣＳリレーションに対して定義されたＬＣＳ組変数の考

え方である。異ったＧＣＳリレーション定義が参照している同一のＬＣＳ変数を、意味的に同一とみる（即ちＧＣＳ定義全体のグローバルな変数とみる）か、単に１つのＧＣＳリレーション内のローカルな変数とみるかの差である。我々は、前者の立場を取ることにする。後者の目的に用いる為には、ＬＣＳ変数名を変える必要がある。この時の正しい解は、第１及び第２の解である。第２の解が、意味的には正しいが、この様に、ＬＣＳ変数をユーザに意識させるかが問題となる。ここでは第１の解に基づいて検討することとし、その後、第２解への拡張を試みることとする。

図2.5のＧＣＳ問合せは、図2.4のＧＣＳリレーション定義式を用いて、図2.7の様な、全体ＬＣＳ問合せに修正される。図2.8は、図2.7の全体ＬＣＳ問合せの問合せグラフ〔ＴＡＫＩＭ８０、ＷＯＮＧＥ７７〕である。図2.8内で、

```
range (e,EMP：1)(r,REP：3);
range (pe,PRUJ－EMP：4)(di PROD：2)(j,SUBJ：2)
retrieve into RSLT(e.ename,pe.pname)
where
      e.@E＝e.@E and
      j.subjectn＝"DISTRIBUTED DATABASE
                                  SYSTEMS" and
      e.@R＝r.@R and pe.@E＝e.@E and
      pe.@P＝d.@P and j.@P＝pe.@P;
```

図2.7　図2.6の全体ＬＣＳ問合せ

結合ｊ．@Ｐ＝ｄ．@Ｐは、図2.7の条件式には現われない。しかし、これは結合式ｊ．@Ｐ＝ｐｅ．@Ｐ and ｐｅ．@Ｐ＝ｄ．@Ｐから推移関係によって導出できる。又、問合せの条件式ｅ．@Ｅ＝ｅ．@Ｅは除かれる。

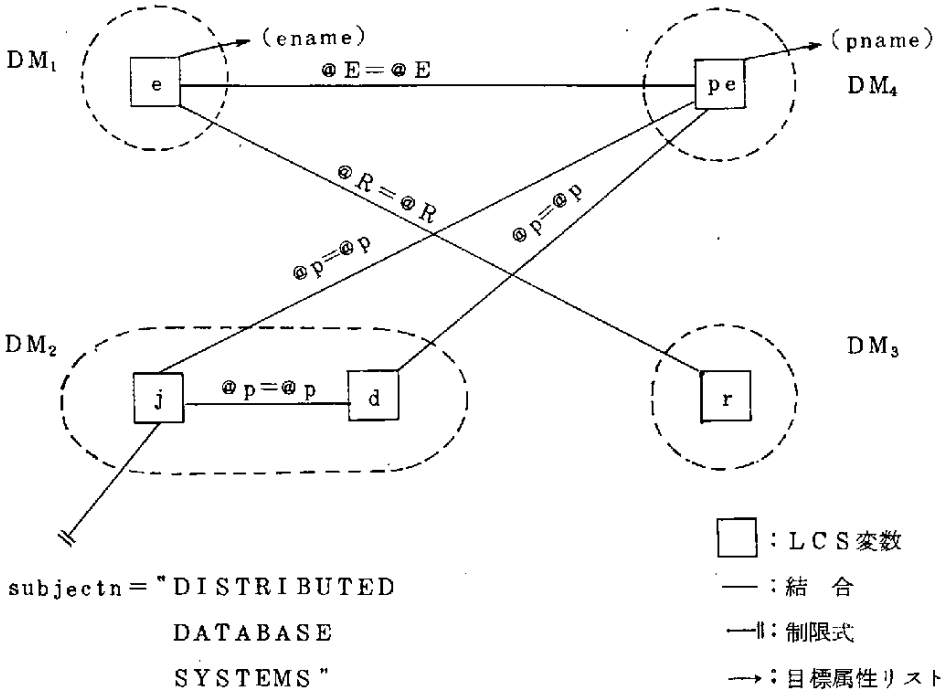


図2.8　図2.7の問合せグラフ

2.2.2　分散問合せ処理

表現変換によって生成された全体ＬＣＳ問合せ（ｇｌｏｂａｌ ＬＣＳ ｑｕｅｒｙ ｏｒ ＧＬＱ）は、各データモヂュール（ＤＭ）内のＬＣＳリレーションを参照している。従って、全体ＬＣＳ問合せを処理する為には、各ＤＭでのローカルな処理と、ＤＭ間でのリレーション の通信とが必要となる。この一連のＤＭでのローカルな処理、及びＤＭ間での通信の集合を、分散問合せ処理（ｄｉｓｔｒｉｂｕｔｅｄ ｑｕｅｒｙ ｐｒｏｃｅｓｓｉｎｇ ｏｒ ＤＱＰ）と呼ぶ。

分散問合せ処理（ＤＱＰ）は、以下の２つの処理から成る。

(1)　初期ローカル問合せ処理（ｉｎｉｔｉａｌ ｌｏｃａｌ ｑｕｅｒｙ ｐｒｏｃｅｓｓｉｎｇ ｏｒ ＩＬＱＰ）

(2)　ＤＭ間通信処理

(1)のＩＬＱＰは、各ＤＭで独立にローカルに行われる問合せ処理である。(2)のＤＭ間通信処理では、複数のＤＭが、通信ネットワークを用いて、互いに通信し合いながら、ＤＭ間にまたがった要求（ｉ.ｅ.. ＤＭ間のリレーション結合）が処理される。

2.2.2では、初期ローカル問合せ処理（ＩＬＱＰ）を論じ、2.2.3ではＤＭ間通信処理について論じる。

2.2.3 初期ローカル問合せ処理

初期ローカル問合せ処理（ＩＬＱＰ）は、全体ＬＣＳ問合せ（ＧＬＱ）内で各ＤＭによって独立して処理出来る部分の処理を行うことである。この例としては、ＤＭ内のＬＣＳリレーションに対する制限（ｒｅｓｔｒｉｃｔｉｏｎ）、射影（ｐｒｏｊｅｃｔｉｏｎ）、結合（ｊｏｉｎ）の処理である。これによって、ＤＭ間で行わなければならない問合せ処理の為の通信処理量を減少できる。

例えば、図2.8を考えてみよう。この図において、各DM_iで以下の処理がローカルに可能である。

DM_1 : R_1 ← ＥＭＰ〔＠Ｅ，＠Ｒ，ｅｎａｍｅ〕；

DM_2 : R_2 ← ＳＵＢＪ〔ｓｕｂｊｅｃｔｎ＝"ＤＩＳＴＲＩＢＵＴＥＤ ＤＡＴＡＢＡＳＥ ＳＹＳＴＥＭＳ"〕〔＠Ｐ＝＠Ｐ〕ＰＲＯＤ；

R_2 ← R_2〔＠Ｐ〕；

DM_3 : R_3 ← ＲＥＰ〔＠Ｐ〕；

DM_4 : R_4 ← ＰＲＯＪ－ＥＭＰ〔＠Ｅ，＠Ｐ，ｐｎａｍｅ〕；

DM_1では、ＤＭ間の結合に必要な属性＠Ｅと＠Ｒの値と、出力属性（目標属性）ｅｎａｍｅが必要なだけである。DM_2では、ＳＵＢＪの制限（ｓｕｂｊｅｃｔｎ="ＤＩＳＴＲＩＢＵＴＥＤ ＤＡＴＡＢＡＳＥ ＳＹＳＴＥＭＳ"）と、ＰＲＯＤとの＠Ｐについてのローカルな結合とを行う事が出来る。DM_3とDM_4に於いても同様なローカル処理を行える。このローカル処理の結果、問合せは、図2.9の様になる。

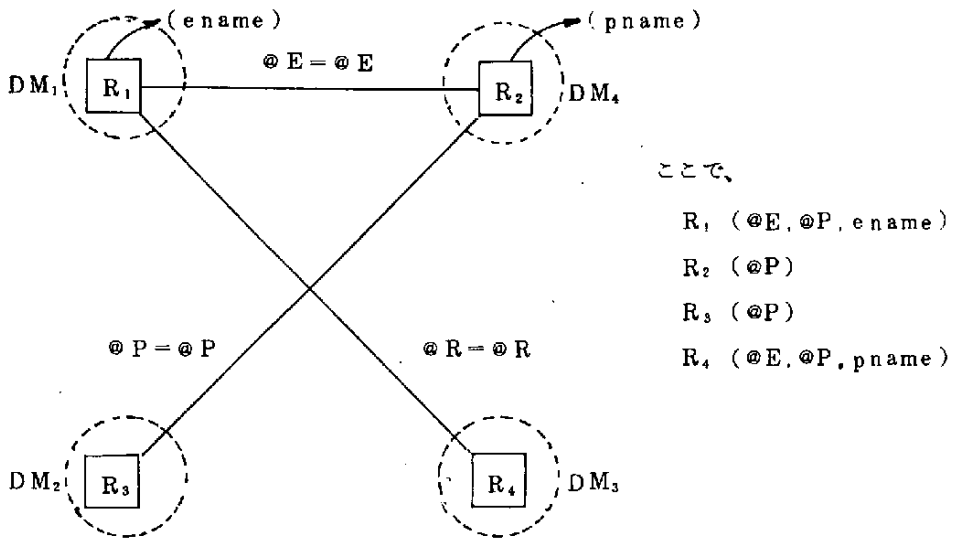


図2.9 ＩＬＱＧＰ後の問合せグラフ

従って、図2.8の全体ＬＣＳ問合せから、各DM_iで初期ローカル問合せ処理（ＩＬＱＰ）を実行する為に図2.10の様なＩＣＳ問合せが生成され、各ＤＭで実行される。

```
(1) DM1   range (e,EMP);
          retrieve into R1 (e.@E, e.@R, e.ename);
(2) DM2   range (j,SUBJ)(d,PROD);
          retrieve into R2 (j.@P)
          where
                j.@P=d.@P  and
                j. subjectn="DISTRIBUTED DATABASE
                                          SYSTEMS";
(3) DM3   range(r,REP);
          retrieve into R3 (r.@R);
(4) DM4   range (pe,PROJ-EMP:4);
          retrieve into R4 (pe.@E, pe.@P, pe.pname);
```

図2.10　ＩＬＱＰの為のＬＣＳ問合せ（ＬＱ）

異種のデータベースシステム（ＤＢＳ）を、ＤＤＢＳがサポートする場合に、初期ローカル問合せ処理（ＩＬＱＰ）は重要となる。ＣＯＤＡＳＹＬ　ＤＢＳ，ＩＭＳ　ＤＢＳといった構造型のＤＢＳは、リレーショナルDBSと異なり、閉包性（closure property）を有しない。この為、データベースから導出された結果を、再度、同一のアクセス言語によってアクセス出来ない。構造型データベースシステムでは、データベースはある構造（e.g.ネットワーク構造）の下でのみ、値の存在が許される。しかし、この構造に基づいて導出された結果は、単に値の集合であって、初めのデータベースの構造とは独立となってしまう。従って、導入された結果を、基の構造によってアクセスする事は出来ない。

ＤＭ間の通信処理では、ある演算の結果から、他の結果を導出する処理が必要となり、閉包性が必要となる。このことより、ＣＯＤＡＳＹＬ型のＤＢＳでは、ＤＭ間の通信処理のサポートを行えない。即ち、分散型データベースシステム（ＤＤＢＳ）では、リレーショナルＤＢＳ以外の構造型ＤＢＳは、単にベースのデータの保存器と考えられる。分散問合せ処理を実行する為には、まず、構造型の異種ＤＢＳから、ＤＭ間での通信処理に必要なデータを導出せねばならない。この導出は、各ＤＭの初期ローカル問合せ処理（ＩＬＱＰ）によってなされる。

以上より、ＩＬＱＰでは、以下の処理がなされる必要がある。

(i) 共通言語ＱＵＥＬによるローカルＬＣＳ問合せを、各ＤＢＳで実行可能なアクセス言語によるプログラムに変換する。

(ii) このプログラムを実行させる。

(iii) 結果を、リレーションとして、出力する。

この結果は、各ＤＭのリレーショナル作業領域（ＲＷＳ）に格納される。

図2.11はＩＬＱＰの処理概要を示している。各ＤＭの全体データベースプロセッサ（ＧＤＰ）は、ＩＬＱＰの為のＬＣＳ問合せ（ＬＱ）を受け取ったならば、これをローカルデータベースプロセッサ（ＬＤＰ）にわたす。ＬＤＰは、ＬＱをそのＤＢＳで実行可能な言語、例えば、COBOL ＤＭＬに変換し、ＤＢＳを起動して実行させる。その実行結果は、リレーションとして、リレー

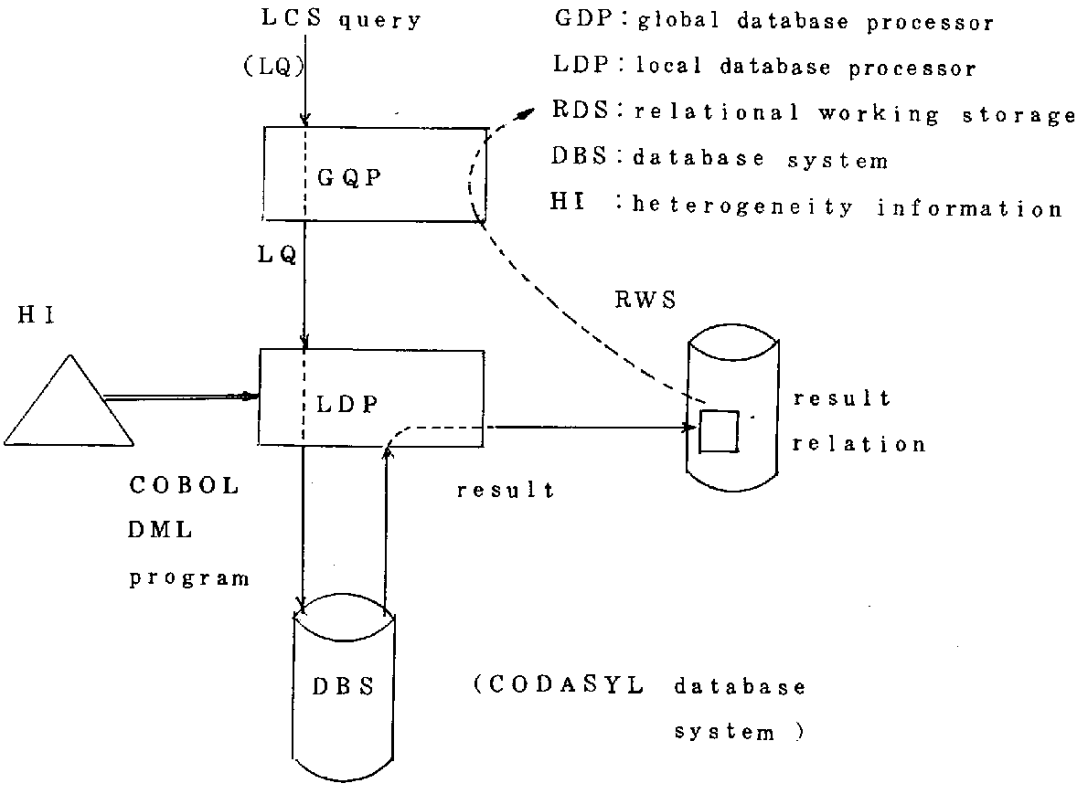


図2.11　ＩＬＱＰの概要

ショナル作業領域（ＲＷＳ）に格納される。ＲＷＳ内のリレーションは、ＧＤＰ間の通信処理の為に利用される。ＬＤＰについては、第３章について詳論する。

同種システムにおいては、ＬＤＰは、そのＤＢＳであり、ＲＷＳもそのＤＢＳ内に設ける事が出来る。

例えば、DM_1は、3.5.1で述べるＣＯＤＡＳＹＬ型のＰＲＤＢＳであるとしよう。PRDBSに対して、ＬＣＳリレーションＥＭＰは、3.5.1に述べるリレーション上の図2.12に示すビューであるとする。このＥＭＰに対する図2.10(1)のＬＣＳ問合せは、ＬＤＰによって、

```
range (e,EMPLOYEE)(er,EMP−REP−LNK);
range (r,REPORTR);
define EMP(e, @E, e.eno, e.endme,
                e, age, e.birth−place, r.@R)
where
      e.@E=er.@E and er.@R=r.@R;
```

図2.12 DM_1 のLCSリレーション

COBOL DMLに変換され、PRDBS上で実行され、結果が、R_1 リレーションとして、DM_1 のRWS_1 に格納される。

2.2.4. DM間通信処理

本項では、ILQP後のDM間にまたがったリレーション間の結合演算を、通信ネットワークを用いたDM間での通信と、各DMでのローカル処理によって、即ち、通信処理によって実行する問題について論じる。我々は、この処理を、DM間通信処理と呼ぶ。DM間の通信処理は、次の4つの機能モヂュールによってなされる〔図2.13〕。

(i) 全体データベースプロセッサ(GDP)

(ii) リレーショナル作業領域(RWS)

(iii) 分散情報(DI)

(iv) 通信ネットワーク(CN)

RWSは、そのDMでの初期ローカル問合せ処理(ILQP)の結果リレーション、DM間通信処理の中間結果リレーション、他のDMから転送されてきたリレーションを格納する為の作業領域である。RWS内のデータは、全て、リレーション形式である。

GDPは、RWS内のリレーションに対して、結合(join演算)を行い、その結果を、再びリレーションとしてRWSに格納する〔図2.14〕。GDPは、RWS内のリレーションの操作管理を行うことの出来るリレーショナルDBMSと考えることができる。GDPは、次のリレーショナル演算を行うことが出来る。

(i) RWS内の2つのリレーションR_1 とR_2 との間の結合(join)

(ii) RWS内のリレーションRの属性aについてのソート

(iii) 受信しているリレーションRを属性aについてソートしながらRWSに格納

RWS内のリレーションは、動的に生成、消滅されていく。これらは各DMの分散情報(DI)内に、そのスキーム及び、パフォーマンス情報が格納される。GDPは、データモヂュール(DM)間での、通信処理のスケジュール、分散実行の制御も行う。

GDP : global database processor
DI : distribution information
RWS : relational working storage
CN : communication network

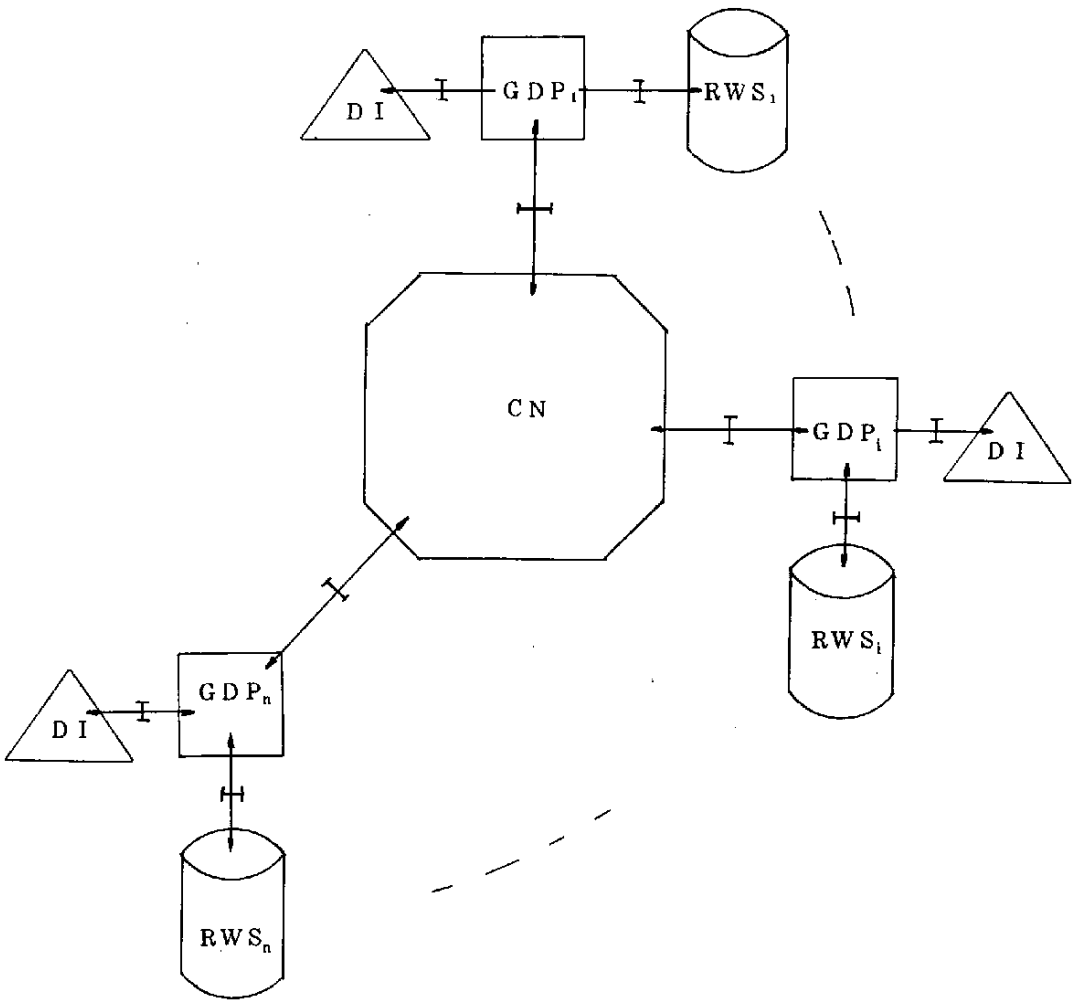
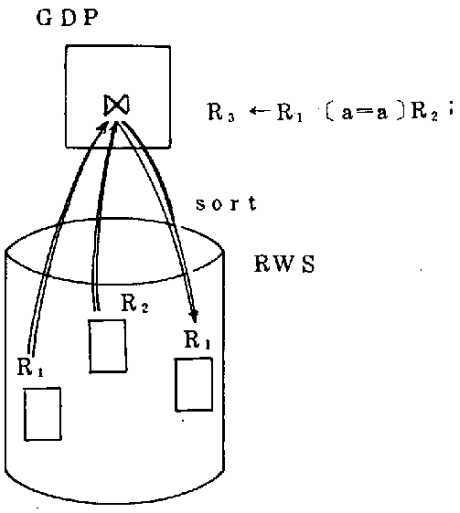
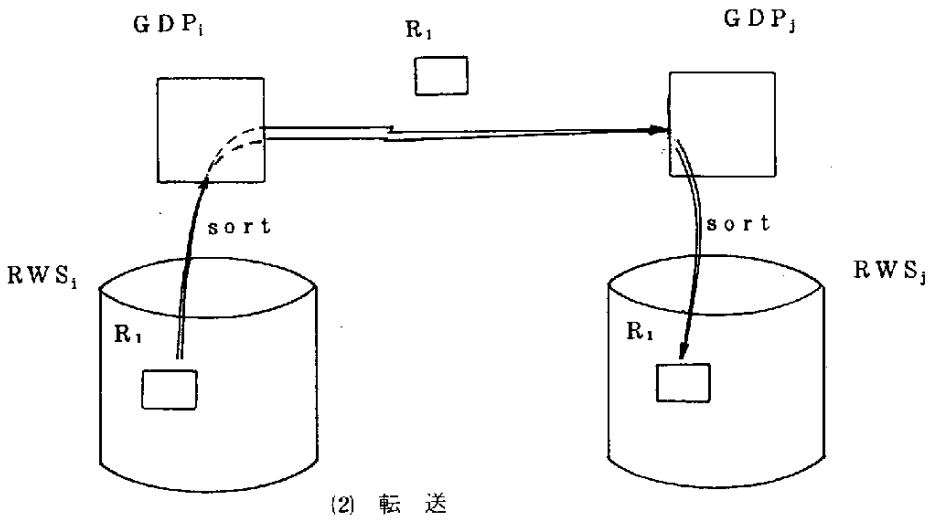


図2.1.3 DM間通信処理アーキテクチャ

スケジュール、分散制御に必要な情報も、DI内に格納されている。スケジューリング、制御方法としては、次の可能性がある。



(1) 結合処理



(2) 転 送

図2.1.4 GDPとRWS

(i) スケジュール　　動的決定法か静的決定法か

(ii) 制　　御　　分散制御か集中制御か

これらについては、後述する。

分散情報（ＤＩ）は、ＧＤＰの処理に必要となる情報が格納されている。

ＤＩの情報としては、以下の種類がある。

(i) ＧＣＳ及び各ＬＣＳのスキーマ情報（論理データ構造の記述）

(ii) ＧＣＳとＬＣＳとの対応情報（ＧＣＳ定義情報、所在情報）

(iii) 各ＬＣＳリレーションのパフォーマンス情報

（ｅ.ｇ. カーディナリティ、属性の選択度）

(iv) ＤＭ間通信処理の中間結果管理情報（ＲＷＳ内リレーションの情報）

（スキーマ、パフォーマンス情報）

これ等の情報は、全てリレーションとして管理される。ＧＤＰにとって、ＤＩは、ＲＷＳ内のリレーションと同様なアクセス方法（ｅ.ｇ.ＱＵＥＬによるアクセス）でアクセスされることができる。

通信ネットワークは、ＧＤＰ間で、リレーション及び制御情報（ｅ.ｇ.コマンド，ａｃｋｎｏｗｌｅｄｇｅｍｅｎｔ）を、転送する為に用いられる。通信ネットワークは、発進ＤＭと目的ＤＭ(s)間で、高信頼でかつトランスペアレントな通信を提供することが出来るとする。分散型データベースシステム（ＤＤＢＳ）に於けるＤＭ間通信処理に於いて、どの様な通信形態を物理的にサポートするかが重要である。通信形態としては、以下の２つがある。

(i) １対１ネットワーク（ｐｏｉｎｔ−ｔｏ−ｐｏｉｎｔ ｎｅｔｗｏｒｋ）

(ii) １対ｎネットワーク（ｂｒｏａｄｃａｓｔ ｎｅｔｗｏｒｋ）

(i)は、ＡＲＰＡＮＥＴ，ＤＤＸの様な広域パケット交換ネットワークにおける通信形態で、２つの通信実体間での通信をサポートしている。一方、(ii)では、１つの通信実体から出されたメッセージが、同時に複数の実体で受信出来る。これは、同軸ケーブルの様な通信媒体を共有した形のローカルネットワーク又は無線ネットワークでサポートされている。

2.2.5 通 信 形 態

通信ネットワークとしては、以下の２種がある。

(i) １対１通信ネットワーク

(ii) １対Ｎ通信（放送）ネットワーク

(i)は、ＡＲＰＡＮＥＴ、ＣＹＣＬＡＤＥＳ等の広域パケット交換ネットワークであり、２つの通信実体間のトランスペアレントな通信手段を提供している。通信速度は５０ｋｂｐｓ程度の中へ低速なものである。あるデータ量ｘをサイトｉからｊに転送する通信コストを$C_{ij}(x)$と表すと、これは、(1)式の様に表せる〔ＨＥＶＮＡ７８〕。

$$C_{ij}(x) = a + b \cdot x \quad \cdots\cdots(1)$$

ここで、ａとｂは定数である。ａは、通信の初期化コストである。(1)は、通信コストが、サイト間の距離に独立で、量のみに依存する事を示している。又、通信コストは、通信時間とする。ネットワークは、低負荷で、各ネットワークノード（ＩＭＰ）での待ち行列遅延は存在しないものとしている。

一方、(ii)は、Ethernet、無線通信ネットワークの様に、通信媒体（同軸ケーブル、電波）を各サイトが共有しているネットワークである。即ち、1つのサイトの発したメッセージは、ネットワーク上の全てのサイトによって同時に受信できる。この時、サイトiからn個のサイト $j_1, \cdots, j_n$ に、量xを転送するコストを $Ci\{j_1, \cdots, j_n\}(x)$ とすると(2)式の様に、表せる〔TAKIM81〕。

$$Ci\{j_1 \cdots j_n\}(x) \triangleq a + b \cdot x \cdots\cdots(2)$$

即ち、通信コストは、サイト間の距離、転送目的サイト数に独立で、量xにのみ依存する。(i)のネットワークでは、(3)の様になってしまう。

$$Ci\{j_1 \cdots j_n\}(x) \triangleq n \cdot (a + b \cdot x) \cdots\cdots(3)$$

この1対n通信機能を用いると分散問合せ処理を有効にかつ簡単に行える〔TAKIM81, 82b〕。この方式については、新ためて述べることにする。ここでは、(i)のネットワークについて考えることにする。

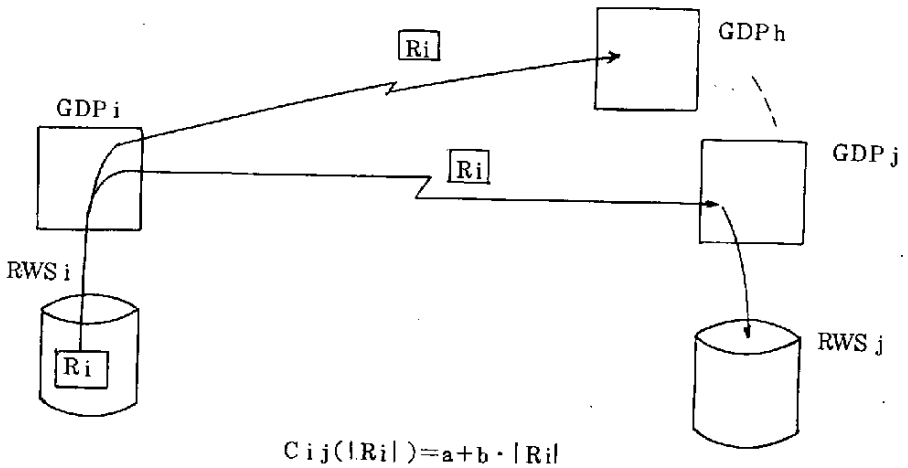
(i)は、低速であることから、各サイトでの処理コストは、通信コストと比較して無視し得るものとする。

分散型データベースシステム（DDBS）に於いて、1対N通信は、以下の点から有効である。

(i) あるデータモヂュール（DM）から発せられた制御／状態情報を他の全てのDMが受信できる。この為、各DMは、他の全てのDMの状態を知り得ることになり、各DMが、全て、マスタコントローラとして"独自に、かつ同一"の決定を行うことができる。従って、完全な分散制御が可能となる。

(ii) DM間の結合処理に於いて、1つのDMの転送したリレーションは、これと結合すべきリレーションを有している全てのDMによって受信できる。又、半結合（semi−join）手法に於いて、リレーションの結合属性についての射影に対する応答として、bit−mapを放送できる。各DMは、bit−mapによって結合処理を行える。

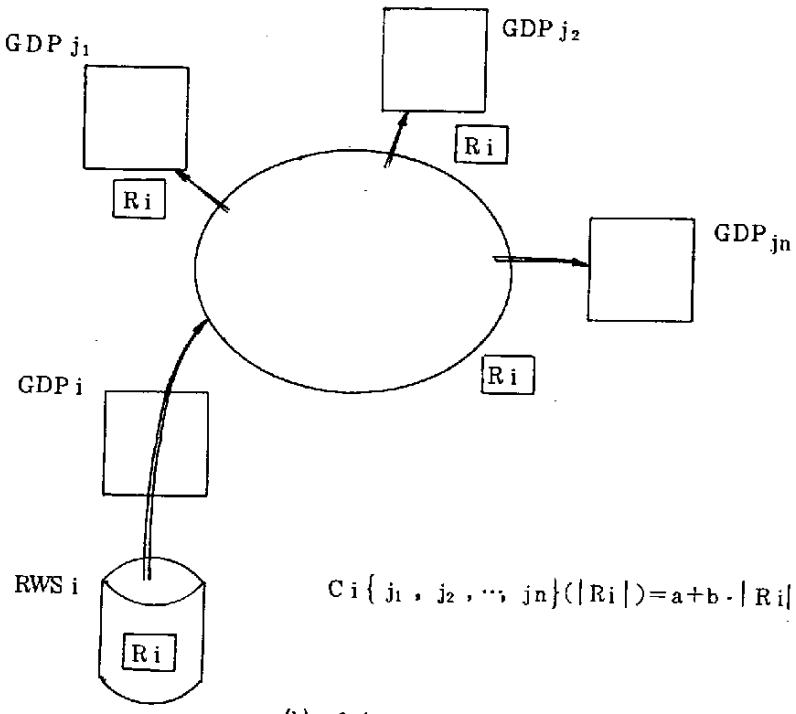
(iii) 原子的な更新オブジェクトが複数のDMにまたがって存在している時、これらのDMに対して、同時にupdate命令を物理的に届ける事ができる。冗長コピーに対しても、同様である。これによって、DDBSに於ける同時実行制御を容易にすることができる。



$$C_{ij}(|R_i|)=a+b\cdot|R_i|$$

$$C_{i\{j,h\}}(|R_i|)=2\cdot(a+b\cdot|R+|)$$

(a) 1 : 1 network



$$C_{i\{j_1,j_2,\cdots,jn\}}(|R_i|)=a+b\cdot|R_i|$$

(b) 1 : n network

図2.15 通 信 形 態

2.2.6 中間結果の見積り方法

見積りは、関係のカーディナリティ（組の数）と、属性の選択度（selectivity）によってなされる。Rを関係とし、aをその属性する。Aを、属性aの定義域とする。属性aの選択度slRaは、次の様に定義される。〔HEVNA78，SELIP79〕。

$$slRa \triangleq card(R〔a〕)／card(A) \quad \cdots\cdots(1)$$

ここで、集合をSとすると、card(S)はSのカーディナリティである。

R_1 とR_2 を2つの関係とし、aをこれらの間の結合（e.g. $R_1.a = R_2.a$）した場合の結果の見積りについて考える。まず、次の仮定を設ける。

(i) 各属性の値の分散は、各々の関係内で均一分散している（正確には、aはR_1 とR_2 内のどちらか一方で均一分散していればよい）。

(ii) 関係内の属性間の値の分散は、互いに独立である。

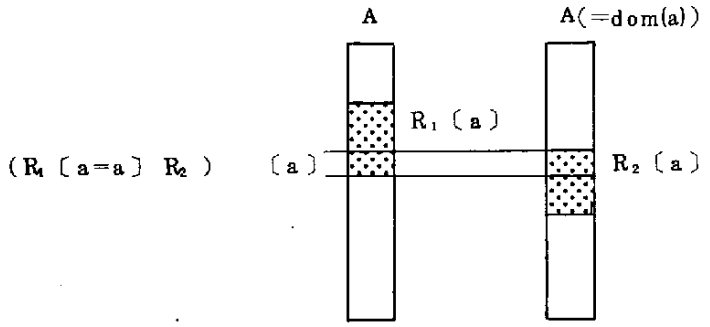


図2.16 選 択 度

この時、結合結果をR（$=R_1$〔a＝a〕R_2）とするとRは次の様に見積もれる。

$$\begin{aligned} card(R〔a〕) &= card(R_1〔a〕)\cdot slR_2 a \\ &= card(R_2〔a〕)\cdot slR_1 a \\ &= card(A)\cdot slR_1 a \cdot slR_2 a \\ &= card(R_1〔a〕)\cdot card(R_2〔a〕) \\ &\qquad card(A) \\ slRa &= slR_1 a \cdot slR_2 a \end{aligned}$$

$$\cdots\cdots(2)$$

2.2.7 DM間・結合処理方法

異なったデータモデュール（DM）に存在するリレーション間の結合（join）処理方法について述べる。まず、RとSを各々DM_1 とDM_2 に存在するリレーションとする。aをRとSの結合属性とする。等結合R.a=S.aの処理について考える〔図2.17〕。この結合を処理する為には、一方のリレーションを他に転送する必要がある。

1つの方法は、一方のリレーション、例えばRを、DM_2 に転送し、DM_2 で結合を行う〔図2.

18(a)〕。この時の通信コストCjは、リレーションRの量 |R|となる。

$$C_j = |R| \quad \cdots\cdots(1)$$

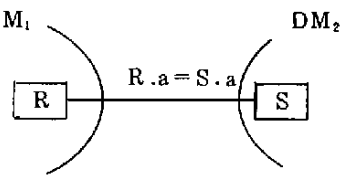


図2.17　DM間結合

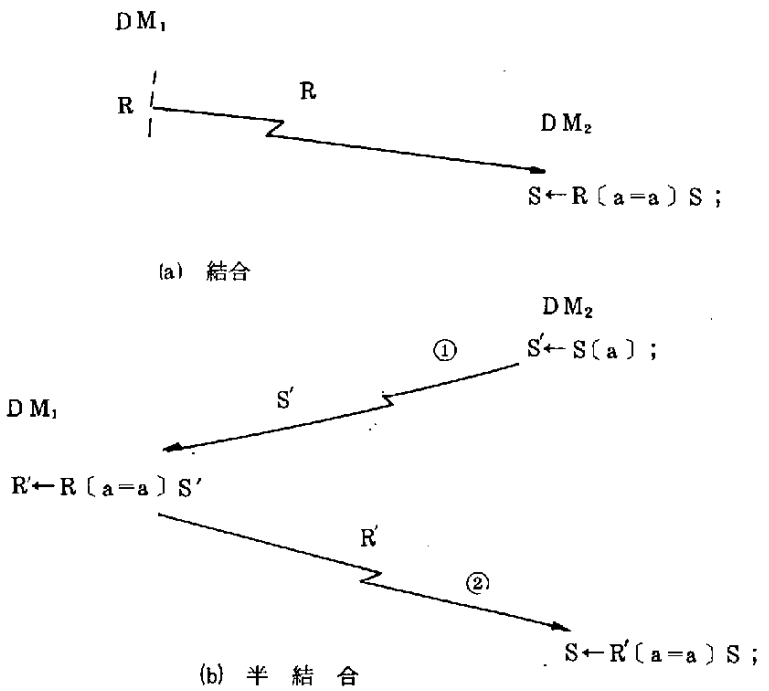


図2.18　結合と半結合

これに対して、半結合〔BERNP79〕（semi−join）と呼ばれる手法では、まず、リレーションSの結合に必要な部分、S〔a〕（Sのaについての射影）（＝S′）を、DM2からDM1に転送する。DM1では、S′とRとの結合を行い、その結果（R′）を、DM2に転送する〔図2.18(b)〕。DM2では、R′とSとの結合を行い結果を得る。この時の通信コストをC_Sとすると(2)式の様になる。

$$C_S = |S'| + |R'| \quad \cdots\cdots(2)$$

半結合は、S〔a〕によって、リレーションRを制限してから、転送させるものである。

ここで、属性aに対してdom(a)と|a|は、各々その定義域その長さを表すものとする。リレーションAに対して、|A|、card(A)は、各々、Aのサイズ、カーディナリティを表すものとする。又、^{w}Aは、Aの組の長さ、i.e.,$|A| = card(A) \cdot {}^{w}A$, とする。関係Aに於ける属性aの選択度（selectivity）〔HEVNA78〕をslAaと記すことにする。

まず、C_sについて考える。

$$|S'| = |dom(a)| \cdot sl_{sa}$$

$$|R'| = |R| \cdot sl_{sa}$$

よって、

$$C_s = sl_{sa}(|dom(a)| + |R|)$$

半結合が、結合に対して有利となる為には、$C_s \leq C_j$でなければならない。

$$C_s \leq C_j$$

$$sl_{sa}(|dom(a)| + |R|) \leq |R|$$

$$|R| \geq \frac{sl_{sa}}{1 - sl_{sa}} \cdot |dom(a)|$$

$$|R| \geq \frac{card(S(a))}{card(dom(a)) - card(S(a))} \cdot |a|$$

$$|R| \geq \frac{a_1}{n - a_1} \cdot |a| \quad \cdots\cdots(4)$$

ここで、 $a_1 = card(S'(a))$

$n = card(dom(a))$

ここで、$|R| \geq card(R) \cdot |a| \geq a_1 |a|$

一般に、$n - a_1 \geq 1$なので

$$a_1 \cdot |a| \geq \frac{a_1}{n - a_1} |a|$$

よって、$C_s \leq C_j$となる。DM間の結合を処理する為には、半結合の方が通信コストを低減できる。

木問合せ（tree query）に対して、半結合は、有効な結合処理方法となる〔BERNP81a〕。木問合せとは、ノードをリレーション、枝を結合述語（等結合）とした時、図2.19の様に、木構造として表せる。ここで、各リレーションは、全て異なったDMに存在するとする。この時、図2.19の問合せは、1対1ネットワークのもとでは、次の様にして処理出来る。

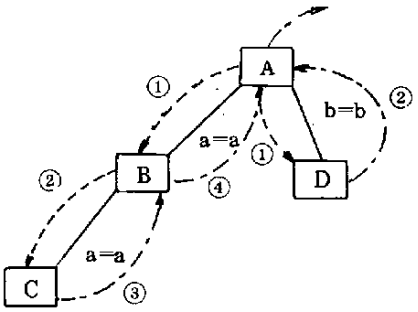


図2.19　木問合せと半結合

(1)　A〔a〕(＝A_i)をBに転送する。同時に、
　A〔d〕(＝A_2)をDに転送する。

(2)　Bで、
　　B←B〔a＝a〕A_1　;
　　B_1←B〔a〕;

(3)　B_1をCに転送

(4)　Cで、
　　C←C〔a＝a〕B_1　;
　　C_1←C〔a〕;

(5)　C_1をBに転送し、
　Bで　B←B〔a＝a〕C_1　;

(6)　BからC_1(＝B〔a〕)をAに転送
　Aで　A←A〔a＝a〕C_1　;

(7)　Dで、A_2を受信したならば
　　D←D〔a＝a〕A_2　;
　　D_1←D〔a〕;

(8)　D_1をAに転送
　　A←A〔a＝a〕D_1　;

この結果、各リレーションの結合属性は、同一の値を持つことになる。半結合で各リレーションを、条件を満足するものに限定した後に、結果を出力するDM(結果DM)に転送する。

この方法は、図2.20に示す様なネットワーク構造を有した一般の問合せに対しては、適用できない。図2.20では結合B・c＝D・cの処理を行う為には、BとDとの間で、結合属性の値の情報の交換が必要となるからである。ネットワーク型の問合せを、半結合で処理する試みとしては〔YOSHI　81〕がある。

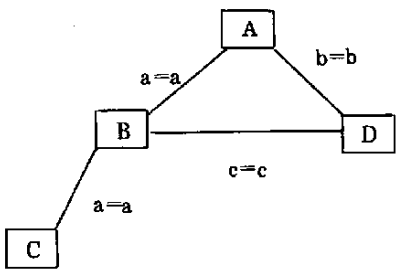


図2.20　ネットワーク問合せ

2.2.8 スケジュールの決定方法

通信処理スケジュールとは、一連のＤＭ間のデータの転送と、ＤＭでのリレーショナル演算の集合である。

非手続的な問合せに対して、可能な通信処理スケジュールは、複数存在する。これらの中から、ある目標を満足する最適なスケジュールを見つけねばならない。

A．目　標

通信処理スケジュール決定の為の目標としては、以下の２点がある〔ＨＥＶＮＡ７８〕。

(1) 全通信処理時間の最少化

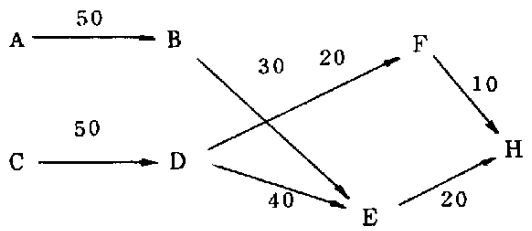
(2) 応答時間の最少化

通信処理時間Ｔは、データの通信コストＣと各ＤＭでの処理コストＰとから成る

$$T = C + P$$

全通信処理時間は、問合せを処理する為に必要な全ての通信と処理の時間の総和である。一方応答時間は、処理を開始してから結果を得れるまでの時間である。一般に、応答時間は、通信とＤＭ処理との並行度を高めることによって、短縮できる。

２つのＤＭ間でのデータの通信コストは、通信される量ｘにのみ依存すると、Ａにおいて述べた。各ＤＭでのローカルな処理時間と、ＤＭ間での通信コストの重みが問題となる。これは、通信ネットワークの伝送速度と、ＤＭの処理能力との相対的な大きさの関連である。広域パケット交換ネットワークは、高々５０ｋｐｂｓの伝送速度しか有しない為に、ローカルな処理時間は、通信時間と比較して無視し得る。従って、通信処理時間は、通信時間、即ち、転送される量にのみ依存する事になる。又、広域パケット交換ネットワークでは、論理的な通信リンクが複数存在し得る為に、１つの問合せの処理に対して、通信を異った通信リンクを通して並行して行える。この為、全通信処理時間と、応答時間とは、一般に異ったものになる。図２.２１は、転送例を示している。ノードはリレーションを表し、矢印は転送を表し、矢印上の数字は転送時間を示している。



全通信処理時間＝２２０

応　答　時　間＝１１０

図２.２１　応答時間と全通信処理時間

ＡとＣの転送、ＢとＤの転送、ＥとＦの転送は、各々並行して行える。

一方、ローカルネットワークは、１０Ｍｂｐｓ～１００Ｍｂｐｓ程度の転送速度を有している。従って、ローカルネットワークは現在の中型機程度のチャネル転送速度を有することから、各

ＤＭでの処理速度と比較し得る程度の通信速度を有することになる。従って、通信処理時間としては、通信時間と共に、各ＤＭでの処理時間も含めて考える必要がある。各ＤＭの処理時間は、処理量に比例するものとする。

我々の考えるローカルネットワークは、同軸ケーブル、無線の様な物理的な共有媒体を複数のデータモヂュール（ＤＭ）によって共有している放送（ｂｒｏａｄｃａｓｔ）ネットワークである。従って、広域パケット交換ネットワークと異なり、ネットワーク内には、同時に最大１つのデータ（ｐａｃｋｅｔ）が存在し得る。この為、通信は、直列に行われることになる。しかし、１つの送出されたデータは、ネットワーク上に結合された複数のデータモヂュール（ＤＭ）によって、ほぼ同時に受信出来る。

他に、ＤＤＢＳの運用上の目標として、以下のものがある〔ＴＡＫＩＭ８０〕

(3) 分散情報（ＤＩ）の最少化と静化

分散情報（ＤＩ）は、全体概念スキーマ（ＧＣＳ）の記述、ＧＣＳとＬＣＳとの対応情報、各ＬＣＳのパフォーマンス情報（ｅ．ｇ．カーディナリティ、選択度）を有している。各ＧＤＰは、分散問合せを実行する為に、分散情報（ＤＩ）を冗長に保持する必要がある。この為ＤＩをなるべく小さくすることによって、格納オーバヘッドを減少させることと、ＤＩ内のデータに対する更新をなるべく減少させること（静化）によって、管理オーバヘッドを減少させることが必要になる。一般に、スキーマ情報（ｉ．ｅ，ＧＣＳとＬＣＳの記述、ＧＣＳと、ＬＣＳとの対応情報）は、ＤＤＢＳのライフサイクルを通してほぼ一定である。一方、カーディナリティ選択度といった、データベースの実現値に関する情報は、データの更新演算によって、変化し得るものである。この為、ＤＩとしては、動的なデータについてはなるべくパフォーマンス情報を不要とすることが望ましい。

Ｂ．スケジュール決定方法

通信処理スケジュールの決定方法には、以下の３種がある。

(i) 静的決定法

(ii) 動的決定法

(iii) (i)と(ii)の混合方法

Ｃ．静的決定法

静的決定法では、通信処理スケジュールは、その実行前に全て決定してしまうものである。通信処理スケジュールは、スケジュール内の各ＤＭ処理において生成される中間結果のサイズを見積ることによって、Ａでのべた全通信処理時間と応答時間を最少とする様に決定される。一般に、ある問合せに対して可能な通信処理スケジュールの数は、数多く存在するので、種々のヒューリスティクスが用いられる。

静的決定法では、ユーザからの問合せを受信したＤＭが、分散情報（ＤＩ）内のカーディナリティ、選択度といったパフォーマンス情報を用いて、以下の様なヒューリスティクスを用いて決定される。例としては、〔ＨＥＶＮＡ７８〕、〔ＷＯＮＧＥ ７７〕等がある。

〔HEVNA78〕のアルナリズムPについて説明する。

アルゴリズムP〔HEVNA78〕

単一結合属性の等結合のみからなるn個のリレーション上の問合せ(これを単純問合せと呼ぶ)に対する通信処理スケジュール決定のヒューリスティクスである。以下に、アルゴリズムPを示す。ここで、n個の異ったサイトにあるR_1, …, R_n を属性aについて結合を取ろうとするとする。C(x)は、量xをあるDMから他のDMに転送する時の通信コストとする。又、通信ネットワークは、1対1ネットワークを用いて、軽負荷であるとする。又ここで、Rを結果リレーションがユーザに出力されるDMとする。

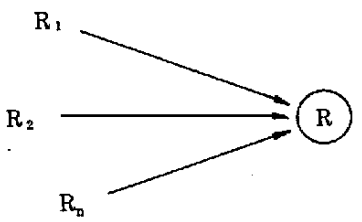
アルゴリズムP

(i) n個のリレーションを、$|R_1| \leq |R_2| \leq \cdots\cdots \leq |R_n|$ となる様に、番号づけをかえる。

(ii) 〔初期解(initial feasible solution or IFS)〕

全てのリレーションに対して

$$r_i \leftarrow C(|R_i|);$$



これは、全てのリレーションを結果DMに集めて、結合をことを示している。

(iii) for i=1, …, n

do begin

for j=1, …, i−1

do

$$r_i' \leftarrow r_j + C\left(|R_i| \cdot \prod_{k=1}^{j} slR_{ka}\right);$$

r_i'を最少とするjを見つける

$$R_j \rightarrow R_i$$

これは、リレーションR_j をR_i に転送することに対応する。

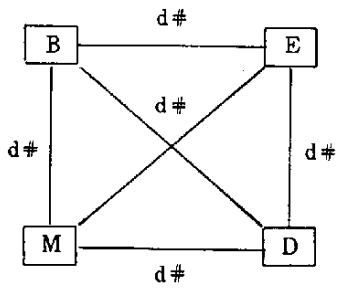
これによって $|R_i| \leftarrow |R_i| \cdot \prod_{k=1}^{j} slR_{ka}$;

$$slR_{ia} \leftarrow slR_{ia} \cdot \prod_{k=1}^{j} slR_{ka} ;$$

スケジュールに、パラレリズムを考慮して転送$R_j \rightarrow R_i$ を加える。

end ;

例えば、図2.2.2の問合せからは、図2.2.3の様な転送スケジュールが生成される。ここで、通信時間は、C(x)＝20＋xとする。



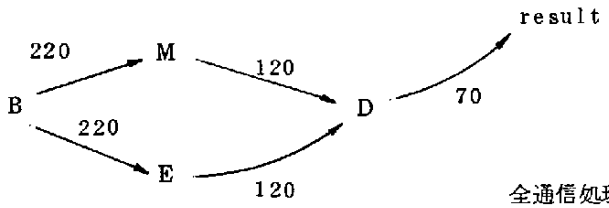
$sIB_{d\#} = 1/5$　　|B|=200

$sIM_{d\#} = 1/2$　　|M|=500

$sIE_{d\#} = 1/2$　　|E|=500

$sID_{d\#} = 1$　　|D|=1000

図2.2.2　問合せ例と選択度



全通信処理時間＝750

応答時間＝410

図2.2.3　アルゴリズムPによるスケジュール

D．動的決定法

動的決定法では、各通信処理の実行情況をモニタしながら、各段階で最適なスケジュールを生成し実行させる方法〔ＴＡＫＩＭ８２ｂ，８２ｃ〕である。この方法を用いる背景には、以下の点がある。

(i)　静的決定法が用いている選択度（ｓｅｌｅｃｔｉｖｉｔｙ）による見積りの確かさに対して疑問がある。選択度では、リレーション内で各属性質が均一分散していることと、各属性の値分散が相互に独立であることの仮定が成り立つことを前提としている。この仮定が、実際のデータベースに於いて、どれだけ成り立つかは、個々のデータベースに依っている。

(ii)　又、各ＤＭ選択度、カーディナリティ等のパフォーマンス情報を、問合せの通信処理スケジュールを決定する為に、保持する必要がある。データベースの更新は、同時に、これらのパフォーマンス情報の更新も必要とする。この為、更新頻度の高いデータベースから成るＤＤＢＳでは、各ＤＭで冗長に保持されているパフォーマンス情報の分散更新によって、互いの一致性を保たねばならない。これは、ＤＭ相互で、パフォーマンス情報の一致性を保つ為の通信オーバヘッドの増大をもたらし、ＤＤＢＳ全体のパフォーマンス低下をもたらす可能性がある。

これらの問題を解決する方法として、パフォーマンス情報を必要としない通信処理スケジュールの動的決定法が考えられている。動的決定方法は、次のステップから成る。

(i) ある通信処理（データの転送と結合処理）の決定

(ii) この処理の結果リレーションについてのパフォーマンス情報（e.g. カーディナリティ、サイズ、選択度）のモニタ

(iii) モニタした情報をもとにした (i)

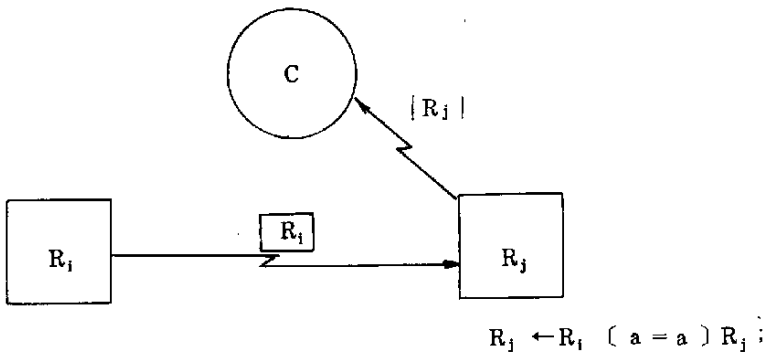


図224 動的決定法の例

図2.24は、動的決定方法の例〔TAKIM82C〕を示している。あるコントローラCは、リレーションR_iをR_jに転送する事を指示する。R_iがR_jに転送され、R_jとR_iとの結合が行われた後、結果リレーションのサイズ（$|R_j|$）がコントローラCに知らされる。Cは、この情報をもとに、転送コストの一番小さいリレーションの転送を指示する。又、転送の並行度を高める為に、結果がある閾値よりも小さければ、ある条件を満足するDMに、コントローラからの指示なしに転送させてしまう。これによって、応答時間の短縮も、全通信コストの短縮と共に試みる方法である。

この方法の問題点は、中間処理の結果情報をモニタする為の通信オーバヘッドである。1対1ネットワークが用いられる時には、一般にコントローラは、問合せに対して、1つだけ存在することになる。このコントローラと、各DM間での、結果情報及び制御情報の通信のオーバヘッドと、このコントローラがパフォーマンスのボトルネックと成り得る。しかし、1対n（放送）ネットワークでは、各DMの発進したメッセージを他の全てのDMが受信できるので、各DMが独自に決定を行える。この為、動的決定法は、1対nネットワークに於いて、有効な方法となると考えている〔TAKIM81，TAKIM82b〕。

他の問題は、通信処理のあるローカルな局面での最適解の積み重ねが必ずしも、問合せ処理全体から見た場合の最適解とは成り得ない点である。動的決定法が、各ステップ毎に、その時点で最少と思われる転送を考えているので、問合せ処理全体を見わたした決定を行えない。1つの解は、Eに述べる静的決定法との混合形態である。

E．混合方式

他に、静的及び動的決定法の混合形態が考えられる〔ＴＯＡＮＮ８１〕。

この方法は、次の様である。

(i) 先ず、選択度カーディナリティといったパフォーマンス情報を用いて、静的に通信処理スケジュールを決定する。

(ii) このスケジュールに基づいて、通信処理を実行させる。

(iii) 実行途中で、生成された中間結果が、最初の見積りに対するある閾値を越えた場合には、その時点から動的決定法に切り換える。

閾値の設定、静的から動的決定法への切り換えの制御といった実際の運用面での制御が、今後の課題である。

2.2.9 制 御 方 式

各ＤＭ及びＤＭ間での通信処理の分散実行を、どの様に制御するかが問題である。

制御方式としては、次の２つが考えられる。

(i) 集中制御方式

(ii) 分散制御方式

A．集中制御方式

集中制御方式では、各問合せに対して、１つのＧＤＰが、通信処理の分散実行の制御を行う。問合せの入力されたＤＭ（結果ＤＭ）のＧＤＰがこの集中コントローラの役目を持つとする。このＧＤＰを、問合せに対する制御ＧＤＰ（ＣＧＤＰ or coordinate GDP）と呼ぶ。

ＣＧＤＰは、以下の機能を有する

(i) ＣＧＤＰで決定された通信スケジュールに基づいて、各通信処理に対して、その開始を対応するＧＤＰに指示し、その結果の応答を受信する〔図2.2.5〕

(ii) 各ＤＭの障害に対する可能な回復を行う。

(iii) 動的決定法では、(i)の応答（ａｃｋ）にのせられてきたパフォーマンス情報を集積して、これから次の最適な通信処理を決定する。

集中制御方式の利点は、ＣＧＤＰが問合せの実行情況全体を集中してモニタ出来るので、ＤＭの障害に対する回復を容易に制御出来る。

しかし、以下の欠点も持つ。

(i) ＣＧＤＰと他のＧＤＰとの制御情報の通信オーバヘッドが存在する。

(ii) ＣＧＤＰがパフォーマンスボトルネックとなる。

(iii) 並行度が高まらない。

(iv) ＣＧＤＰの障害は、問合せのその後の実行を不可能にしてしまう。

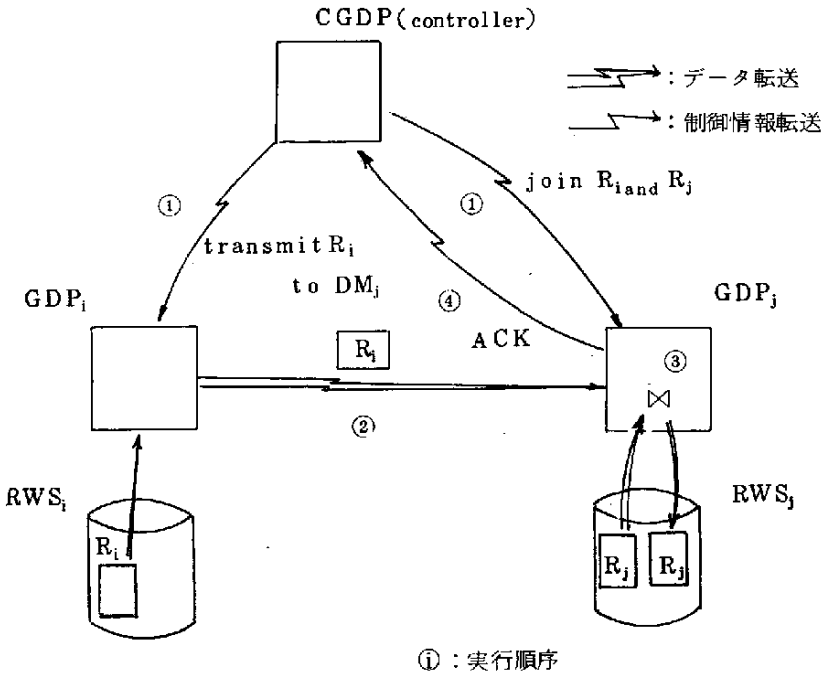


図2.25 CGDPとGDP

B．分散制御方式

分散制御方式では、各GDPが独自の判断によって、通信と処理の起動が行われる。1つの方法は、MEDLEY〔VINED81〕におけるデータフロースキームに基づくものである。

(i) 各GDPに、まず、問合せの静的に決定された通信処理スケジュールにおける各演算をロードする。

(ii) 各GDPは、各演算に対して必要な全ての入力データが揃った時点で通信処理を実行する。

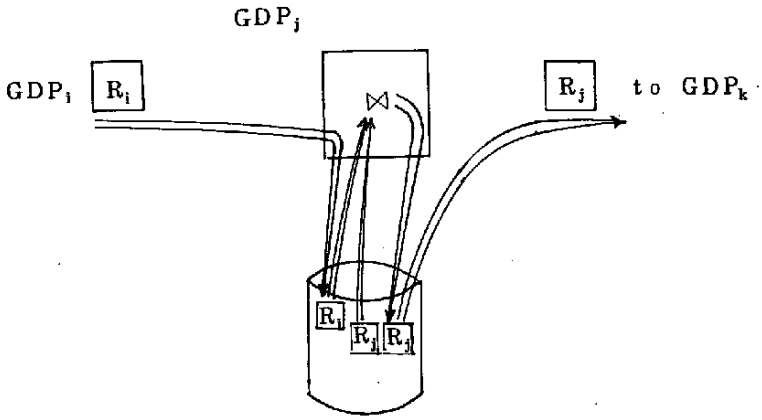


図2.26 分 散 制 御

例えば、図2.26でGDP_jでは結合演算R_i〔a＝a〕R_jは、2つのリレーションR_iとR_jとが揃った段階で起動される。又、この演算が終了したならば、結果リレーションR_jのGDP_kへの転送が起動される。

この様に、分散制御では、Aの集中制御と異なり、CGDPとの制御情報の通信無しに、通信処理を実行できる。この為、通信オーバヘッドが減少すると共に、高い通信処理実行の並行性を達成できる。問題点としては、スケジュール内のあるDMが障害を起こした時、他のDMの実行を中止させたり、障害DMの回復等の制御の困難さがある。

1対1ネットワークの分散制御方式に対して、1対Nネットワークでは、全てのDMが、集中制御に於けるCGDPとなり得ることができる。即ち、あるGDP（e.g。GDP_1）の発進した、そのCDPの状態情報は他の全てのDMによって受信できる。この為、各GDPは、互いに他の全てのGDPの状態を知ることができる為に、同一の決定を行える。この決定で、転送すべきリレーションを有しているGDPは転送（放送）し、有しないGDPはリレーションの到着を待てばよいことになる。全員が、同時に、全体に対する同一の決定を行い、この決定に基づいて各自が行動できる。

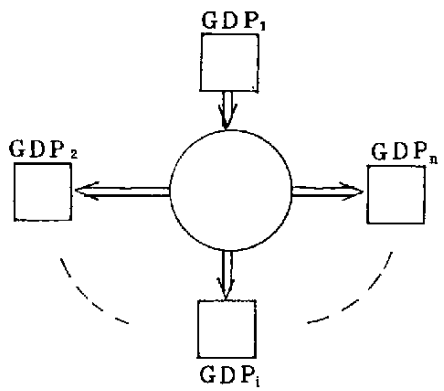


図2.27 1対N network

C．障害管理

問合せの通信処理におけるDMの障害としては、次の2種がある。

(i) 初期ローカル問合せ処理に於けるあるDMの障害

(ii) DM間通信処理に於けるあるDMの障害

ここで、各DMを結合する通信ネットワークは高信頼であって、障害を起こさないものとする。

初期ローカル問合せ処理（ILQP）に於けるDMが障害を起こした場合には、他のDMに冗長コピーが存在しない限り、この問合せの処理の実行を行えない。この為、直ちにILQP中のDMに、処理の中止を連絡し、問合せ処理を停止する。

DM間通信処理では、以下の様にする。

(i) GDP_jで生成されたリレーションR_iをRWS_iに格納する。

(ii) R_iをGDP_jに転送する。

(iii) GDP_jで、結合を行い結果をRWS_jに格納する。

(iv) GDP_jでR_jから生成された後に、GDP_iでR_iの消去を行う。

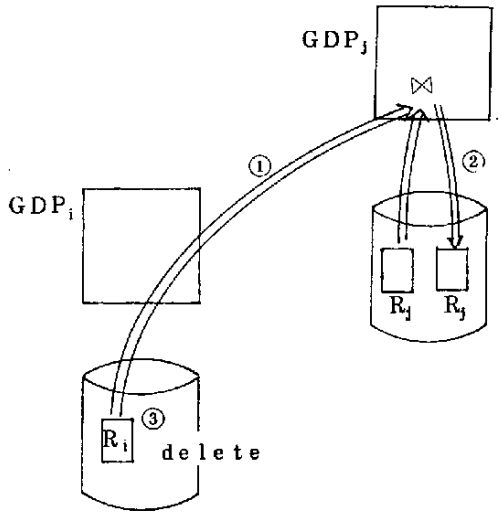


図2.2.8　DM間通信処理

2.2.10　ローカル放送ネットワークに於ける分散問合せ方法

分散型データベースシステム（ＤＤＢＳ）は、複数のデータベースシステム（ＤＢＳ）を、通信ネットワークによって結合し、利用者に対して各システムのｈｅｔｅｒｏｇｅｎｅｉｔｙ，その所在とは独立な全体概念スキーマ（ＧＣＳ）に基づいたアクセスを許すシステムである。ＤＤＢＳでは、ユーザのＧＣＳに基づいたアクセス要求（ＧＣＳ問合せ）は、各ＤＢＳが通信ネットワークを用いて相互に通信し合うことによって処理される。この為に、ＤＤＢＳにおける通信ネットワークの役割は重要である。

ＳＤＤ－１〔ＲＯＴＨＪ８０〕，ＰＯＬＹＰＨＥＭＥ（ＡＤＩＢＭ８０）等の従来のＤＤＢＳは、ＡＲＰＡＮＥＴ，ＣＹＣＬＡＤＥＳ　ｅｔｃ・の広域ｐａｃｋｅｔ－ｓｗｉｔｃｈｉｎｇネットワークを用いて実現されている。このネットワークは、２つの通信実体間のトランスペアレントで高信頼な通信ファシリティを提供している。

これに対して、最近のＥｔｈｅｒｎｅｔ〔ＭＥＴＣＲ７６〕，等のローカルネットワークは、ｃｏａｘｒａｌ　ｃａｂｌｅ等の高速な通信媒体（１－１０ＭＢＰＳ）を複数のサイトで共有することによってｂｒｏａｄｃａｓｔ通信機能（１：ｎ通信）を実現している。即ち、あるｓｉｔｅが送信したメッセージは、ネットワーク上の全てのサイトにおいて、ほぼ同時に受信できる。こうしたローカルネットワーク上に、ＤＤＢＳを実現する試みは、ＳＩＲＩＵＳ－ＤＥＬＴＡ〔ＬＥＢＩＪ８０〕，〔ＴＯＡＮＮ８１〕，〔ＧＵＯＤＭ８１〕等でなされている。

１：Ｎ（ｂｒｏａｄｃａｓｔ）通信機能を用いたＤＤＢＳの問合せの分散処理においては、〔ＴＯＡＮＮ８１〕〔ＧＵＯＤＭ８１〕の試みがある。〔ＴＯＡＮＮ８１〕は、問合せの処理の為の制御情報をｂｒｏａｄｃａｓｔすることによって、完全な分散制御と動的なスケジューリングの実現を試みている。

〔ＧＵＯＤＭ８１〕は、木問合せの処理において、結合値のｂｒｏａｄｃａｓｔ手法を示している。

本論文で述べる問合せの分散処理アルゴリズムは、以下の特徴を有している。

(1) broadcastネットワークによるデータと制御情報のbroadcastと各サイトでのパラレル処理

(2) bit—mapを用いた半結合による通信コストの低減

(3) ネットワーク問合せの処理

(4) スケジュールの動的決定

A．仮　定

DDBSの問合せの分散処理を検討するうえで、以下の仮定を設ける。

(1) 単一ユーザとする。又は、軽負荷であり、互いの問い合せの処理の間で競合はないものとする。

(2) 各DMは、同種な高信頼プロセッサとする。

(3) 各DMは無限の格納エリアを有している。即ち、要求が生じた時は、いつでもデータを、receiveできる。

(4) ローカル処理コストは、転送コストと同程度とする。

(5) 1つの問合せの処理には、同時に1つのリレーションのみを転送する。共同媒体ネットワークなので、1時に1リレーションのみが転送可能できる。

(6) 各DMは、必要な演算をいつでも実行できる。

(7) 等結合のみを考える。

B．分散問合せ処理アルゴリズム

DDBSのGCSは、各DMのLCS上のビュー・リレーションとして定義される。GCS問合せは必要なデータの所在も、各データの格納形態も、不可視なGCSリレーションに対して、QUEL〔HELDG75〕を用いて記述される。GCSはquery modification手法を用いて、LCSリレーションの問合せに（全体LCS問合せと呼ぶ）変換される。この変換は、問合せを入力したDM（これを結果DMと呼ぶ）によってなされる。ここで結果DMは、全体LCS問合せを全DMにbroadcastさせる。受信したDMは、そのDMで処理できる部分の処理を行う（Initial local query processing or ILQP）これによって、通信コストを減少できる。ILQP後の問合せは、結合と目標リストのみから成る問合せである。各DMは、ILQPの終了後、結果情報（cardinality）を他の全DMにbroadcastする。

我々の分散問合せ処理アルゴリズムは、以下の特徴を有している。

(1) データと制御情報のbroadcast

(2) bit—map応答を用いたsemi—join

(3) 通信処理スケジュールの動的決定

(4) 実行の分散制御

C．Broadcasting

ローカルネットワークでは、broadcast通信機能を備えている。この為、あるDM$_i$から転送されたデータは、ネットワーク上の全てのDMが、物理的にほぼ（approaximately）同時に受信できる。この時の通信コストは2.2.5で述べた様に、転送されるデータ量xにのみ依存している。従って、もし転送されたリレーションが複数DM上のリレーションとjoinをとることが必要ならば、関連するこれらのDMが同時にリレーションを受信し、結合演算を並行に行うことができる。

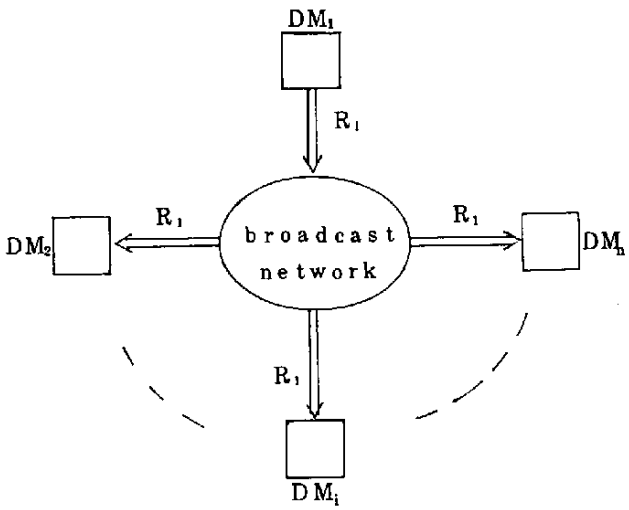


図2.29　broadcasting

又、各DMでの処理結果等の制御情報も、同様にbroadcastされる。このことは、全てのDMが、他のDMの処理状態をモニタできることを意味している。従って、各DMは、これらの情報を基にして、独自にかつ等しい決定を行うことができる。即ち、完全な分散制御が可能であるとともに、各段階での最適なスケジュールを決定する動的スケジューリングも可能となる。

D．Bit-Map Join

DM間のリレーションの結合を処理する為に、半結合手法〔BERNP79，2.2.7〕は、通信量を減少させるうえで重要である。リレーションR_1，R_2，R_3が各々DM_1，DM_2 DM_3に存在し、互いにaによってjoinされているとする〔図2.30〕。この時R_1〔a〕をbroadcastすることによって、R_2とR_3とをrestrictできる。次にR_2とR_3、の結果のどちらかを再びboadcastすれば、R_3には条件を満足するaの値が残ることになる〔図2.31〕。この時R_1〔a〕のコピーは全てのDMに存在するので、R_2〔a〕を転送するかわりにR_2内のaの値のR_1〔a〕に対するbit-mapを転送する方法がある〔図2.32〕。ここで|a|を属性aの長さとし、リレーションに対して|R|はそのサイズ、card

(R)はそのカーディナリティとする。

この時、もし、

$$card(R_2〔a〕)\cdot|a|\geq card(R_1〔a〕)$$

$$\therefore |a| \geq \frac{card(R_1〔a〕)}{card(R_2〔a〕)}$$

ならば、bit-mapの転送の方がコストを低減できる。|a|が4byteとしても|a|=32である。$R_1〔a〕$が$R_2〔a〕$により、32倍大きい時にはじめて値の転送がbit-mapより有利になる。これはほとんどの場合あり得ない。

この各DMが生成されたbit-mapは、結合の応答としてbroadcastされる。従って、各DMは全てのbit-mapを受信し、その積を取れば結合結果を得ることができる。〔図2.32(3)〕。

従って、リレーションR_1，R_2……，Rnが結合属性aによって相互に 結合されているとする。この時、この結合は以下の2段階によって達成できる〔図2.33〕。

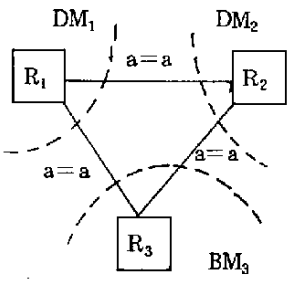


図2.30 結合属性aの問合せ

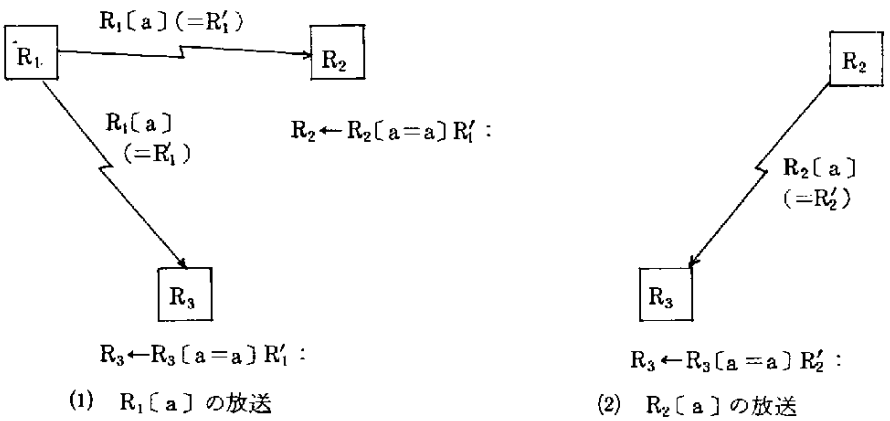
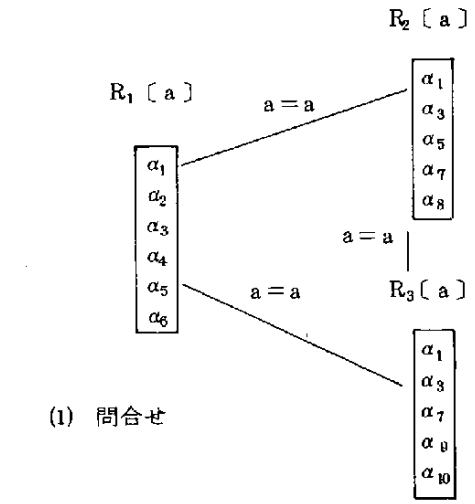
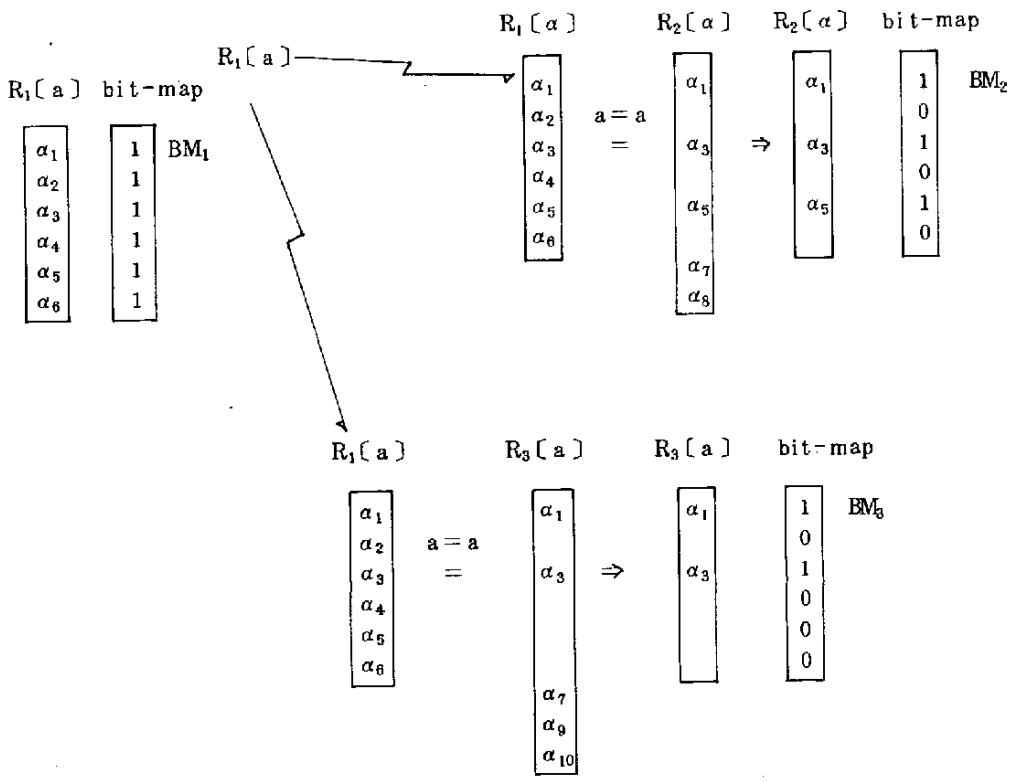


図2.31 図2.30 の処理

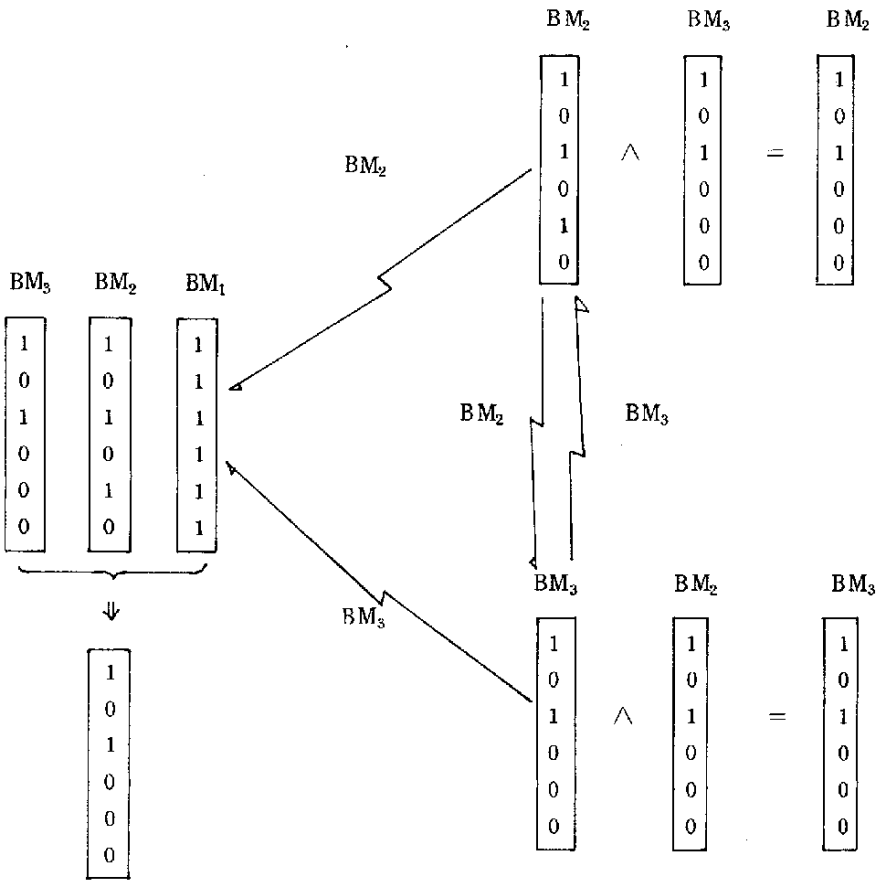


(1) 問合せ



(2) R_1〔a〕の放送と、bit-map の生成

図2.32 bit-mapを用いた半結合



(3) bit-map responseの放送

図2.32 bit-mapを用いた半結合

(1) あるリレーション（R_1 とする）R_1〔a〕（$=R'_1$）のbroadcastを行う。各R_i のでのjoin，$R_i \leftarrow R_i$〔a＝a〕R'_1（i＝2，…，n）を行う。

(2) 各DM_i（i＝2，…n）から、R_i (a)のR_i に対するbit−map（BM_{ia}）のbroadcastと、他のDM_i からのbit−mapの積を取る。

各DM_i は、R_1〔a〕（$=R'_i$）、と全ての他のR_j〔a〕の値をbit−mapとして得れる。この為、各DM_i では全てのリレーションを、集めて結合したのと同じことになる。この為に、(1)及び(2)のステップによって、各DMに生成されたリレーションR_1 …Rnについて、R_1〔a〕$=R_2$〔a〕$=\cdots=$Rn〔a〕となる。

従って、転送コストは、以下の様になる。

$$C = |R_1〔a〕| + (n-1)\cdot card(R'_1)$$

第一項は、R_1〔a〕の転送コストであり、第2項はbit−map応答の転送コストである。1つの結合属性aに関する(1)と(2)の2つの通信処理を合せてaについてのステージ（stage）と呼ぶ。これをst(a)と記す。aをステージ属性と呼ぶ。又、R_1 をst(a)のソースリレーションと呼ぶ。

処理コストについて考える。ここで、bit−map演算処理は、主記憶内で処理できるのでそのコストは無視し得ると考えられる。従って、処理コストは、各DM_i での結合$R_i \leftarrow R_i$〔a＝a〕R_1 のコストが支配的要因となる。ここでR'_1 はaについてソートされており、R_1 はaのインデクスを有して、かつインデクスはソートされているとしている。このため処理コスト，P，は|R|に比例すると考えられる。

$$P = (n-1)\cdot\alpha\cdot|R_1〔a〕|$$

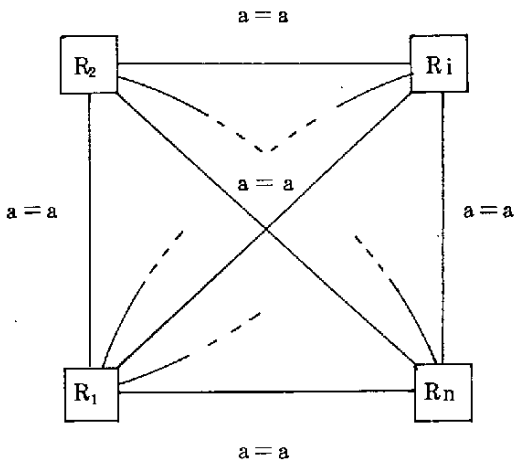
ここで、αは通信コストCに対する重みである。

全通信処理コストTは、次の様になる。

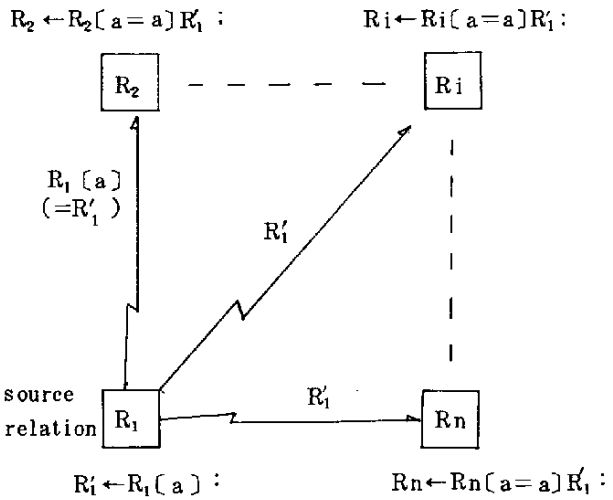
$$T = P + C$$
$$= (1+\alpha(n-1))\cdot|R_1〔a〕|$$
$$+ (n-1)\cdot card(R_1)$$

$\alpha=1$とする（即ち、通信と処理コストは、等価とする）と全通信処理コストは次の様になる。

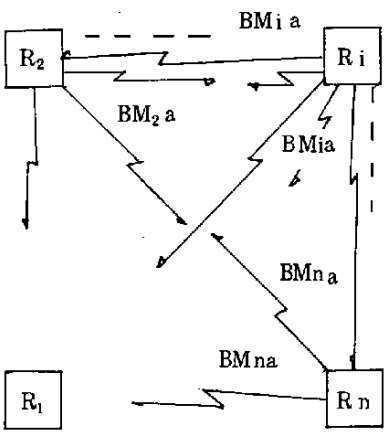
$$T = n\cdot|R_1〔a〕| + (n-1)\cdot card(R_i)$$



(0) 問合せ



(1) R1〔a〕の broadcast



(1) bit-map response の broadcast

図2.33 ステージst(a)の処理

E・スケジュール

我々のアルゴリズムでは、１つの結合属性の処理は、１つのステージによって終了する。従って、m個の結合属性a_1，…a_mからなる問合せは、m個のステージによって処理できる。このステージのシーケンスを問合せ処理のスケジュールと呼ぶ。a_iを現在のステージの属性とする。a_jをこれ以前のあるステージ属性とする。この時、一般にリレーションRが属性a_iとa_jを有しているとする。ソースリレーションR_1とR_2の結合は、R〔a_j〕を制限する。

この為、Rは処理R〔a_j〕のR′に対するbit−mapと共にR〔a_j〕のbit−mapをステージの応答としてbroadcastする。受信したDMは応答のbit−mapと関連する結合属性を有するならばbit−mapによる結合を行う。各DMは属性a_i，a_jについての結合処理後リレーションRから射影R〔a_i〕とR〔a_j〕を生成し、各々a_iとa_jについてソートする。

１つのステージ終了後全てのリレーションの属性a_iについての射影は、同一となる。

R_1，…，R_nを問合せの参照するリレーションとし、各々異ったDM$_1$ …DM$_n$に存在するものとする。a_iを現在のステージ属性とする。STA_{i-1}を、現在のステージst(a_i)以前のステージ属性の集合とする。STA_iは$STA_{i-1} \cup \{a_i\}$となる。各DM$_i$は、リレーションR_iがSTA_{i-1}内の属性b_1を有するならば、st(a_i)の開始時に、R_i〔b_1〕をR_{ib1}として保持しているとする。この時、st(a_i)は、以下の様なstepで実行される

(1) st(a_i)のソースリレーションRkのステージ属性a_iについての射影R_k〔a_i〕(R'_kと記す)がbroad castされる。

(2) 各DM$_j$（j＝1，2，…，n，j≠k）で$R_j \leftarrow R_j$〔$a_i = a_i$〕R'_k；

(3) R_j〔a_i〕のR'_kに対するbit−map(BM_{jai})を生成する。

(4) R_jの属性の中で、STA_{j-1}に含まれる属性（以前のstage属性）の集合をSTA_{ji-1}と記す。STA_{ji-1}の全ての属性b_l（$l = k_j \cdots, h_j$）に対するbit−map(BM_{jbi})を生成する。

(5) 各DM$_j$でステージの応答としてresp$_j$（BM_{jai}，BM_{jbi}，…，$BM_{jb_{kj}}$）をbroadcastできる。resp$_j$が関連しているstage属性集合はST_{j1}である。

(6) 全てのDM$_i$（i＝1，2，…，n）は、他のDM$_p$（p=1，2，…，n，p≠k）からの応答resp$_p$を受信する。R_p属性でSTA_{pi}に含まれる全ての属性b_gについて、bit−mapの積を取る。$BM_{ig} \leftarrow BM_{ig} \wedge BM_{pg}$；

これらのbit−mapをもとにして、条件を満足するリレーションR_pをつくる。

新たに、STA_{p1}内の全ての属性aについてR_{pa}をつくる。

$$R_{pa} \leftarrow R_p \ (a);$$

P_{pa}をaについてソートする。

各DM$_j$はstage(a_i)の処理において、STA_{j1}内の全ての属性各々について他のDM上の値を、全て得ることと同様の処理がbit−mapによって行える。この為、STA_i内の全ての属性値は、各DMにおいて等しくなる。

従って、i第目のstageにおける通信コスト，C_i ，は次の様になる。

ここで、$STA_{j-1} = \{ a_1 , \cdots, a_{i-1} \}$ とする。

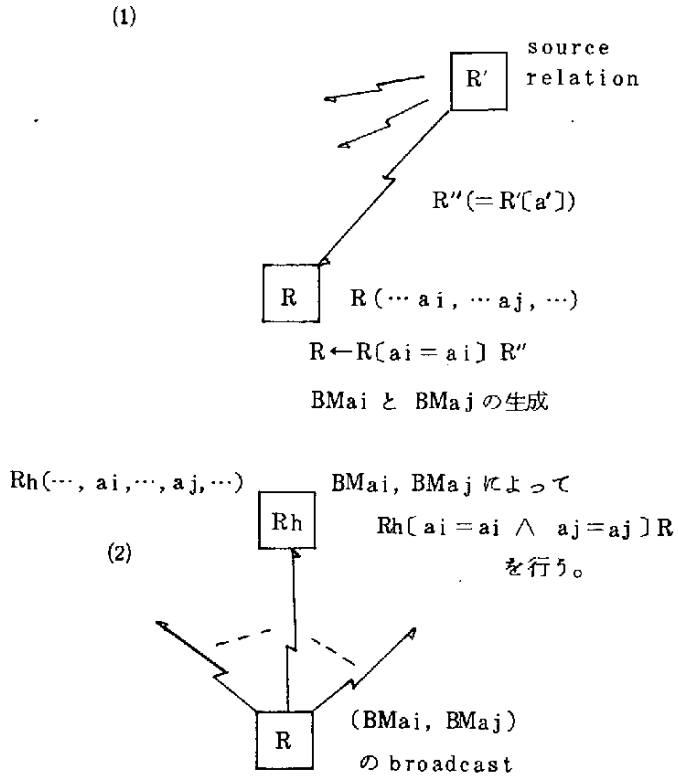


図2.34

STA_{i-1} 内の全ての属性aについてこれを有する全てのリレーションR_i のについての射影 R_i [a]に等しい。これを R[a]と記す。

$$C_i = |R_k[a_i]| + \sum_{\substack{j=1 \\ (j \neq k)}}^{n} \delta ij \cdot \{ card(R_k[a_i]) + \{ \sum_{\substack{l=1 \\ (l \neq k)}}^{i-1} \sigma jl \cdot card(R_j[a_l]) \} \}$$

ここで、

$$\delta 1 = \begin{cases} 1 & \underline{if}\ a_1 \in STA_{ji-1} \\ 0 & \underline{otherwise} \end{cases}$$

$$\delta ij = \begin{cases} 1 & \underline{if}\ a_{1i}\ \ in\ R_j \\ 0 & \underline{otherwise} \end{cases}$$

第1項は、R〔a_i〕のbroadcastするためのコスト、第2項は、そのaのbit-mapのbroadcast costである。第3項は各R_k内が今までのステージ属性を持つ時、そのbit-mapの転送コストである。

処理コストPiは次の様になる。

$$P_i = (n-1) \cdot |R〔a_i〕| \cdot \alpha$$

全処理コストT_iは次の様になる。

$$T_i = P_i + C_i$$

F．ステージ属性の減少

各DM_iは結合属性a_iのステージst(a_i)において応答としてbit-mapを転送しなくて良い場合がある。これは以下の場合である。ここでRをソースリレーションとしてR〔a_i〕がbroadcostされ、R_j←R_j〔$a_i = a_i$〕Rが受信されたDMで行われる。

(1) $R_j〔a_i〕 = R〔a_i〕$の時、$BMja_i$の生成と転送不要である。

即ち、R〔ai〕は、Rjを全くrestrictしていないので、$BMja_i$のbitは全てonである。従って、$resp_j$は、単にpositive acknowledementの役目を持つ。

(2) STA_j内の属性aについて

$R_j〔a〕 = R_{ja}$の時、BM_{ja}は不要となる。a_iの結合によって、R_j内の前のステージ属性aの値には変化がないからである。

(3) ステージst(a_i)終了後、リレーションR_jのみがある属性blを有していて、blは目標属性でない時、R_jからblを消去する。

1つのステージst(a_i)の終了によって、不要となるリレーションとしては、以下のものがある。

(1) R_jは属性a_iのみを有し、a_iは、目標属性ではない時、R_j〔a_i〕のbit-mapを、broadcast後他のackを受信し、結合する必要はない。この為、bit-map転送後、R_jをdeleteする。

$resp_j$として、R_jかdeleteされる情報をのせておく。

(2) R_jは目標属性を有せず、R_jの有する属性はSTA_{i-1}とa_iである時、STA_jとa_iのbit-mapのbroadcast後、R_jをdeleteする。R_jは、後のステージの処理に不要となるからである。

F．結果の生成

問合せは、全結合属性a_i，…，a_mについてn個のstageが終了した時、各DM内のリレーションの結合属性についての射影は等しくなっている。又、結果的に、各DMが有するリレーションは、目標属性を有するリレーションと、これらの間の結合属性だけとなる。従って、最後にこれらのリレーションを、ユーザの結果DMに転送して、最終的な解を生成し出力する。

G．Decision

次に、どの結合属性についてのステージを行い、どのリレーションをソースリレーションとするかの決定が必要である。

この決定方法として1対nネットワークの利点をいかした、以下の様な動的な決定法を用いる。

(1) 各DM_iは、各ステージ終了後、未だステージ処理をされていない結合属性aについてのカーディナリティを応答にのせて、broadcastする。

(2) 各DM_iからのresponse内のbit-mapによる結合後、各DMは一番カーディナリティの少ないR_k〔a_{i+1}〕を見つける。

各DMは、同一のR_kとa_{i+1}を見つける。これを、次のステージのソースリレーションとステージ属性として見る同一のものがある時は、あらかじめ決められたDMの順によって決められる。

(1)は、responseによるbit-map結合前の情報であるので、(2)による選択は、ステージ終了後の最少のものではない。最少のものを得るためには、bit-map join終了後、その結果の情報（join attribute の cardinarity）をbroadcastする方法もある。我々は、この為の処理遅延を省く為に上記の様な方式を用いた。しかし、この解も十分な候補となり得る。

2.2.11 まとめと今後の課題

本節では、分散型データベースシステム（DDBS）に於ける問合せの分散処理方式について論じた。問合せの分散処理方式では、次の3点について考える必要がある。

(i) 通信ネットワークの通信形態

- 1対1ネットワーク
- 1対nネットワーク（放送ネットワーク）

(ii) 通信処理スケジュールの決定方法

- 動的決定法
- 静的決定法

(iii) 分散実行の制御方法

- 集中制御
- 分散制御

通信ネットワークでは、従来の広域パケット交換ネットワークに於ける1対1通信に加えて、近年のEthernet等の通信媒体を共有したローカルネットワークを中心とした1対n通信機能の

利用が、ＤＤＢＳでは重要となってきている。この機能は、2.2.10で示した我々の分散問合せ処理アルゴリズムの様に、問合せの処理を有効とする。これと共に、更新の分散処理に於いても、データ・オブジェクトの集合に対する原子的（ａｔｏｍｉｃ）な更新を、ｉｎｄｉｖｉｓｉｂｌｅに行う為の有効な手法ともなる。

我々の提案する分散問合せ処理アルゴリズムは、以下の特徴を有している。

(i) １対Ｎ通信の利用

(ii) 動的スケジュール決定方法

(iii) 分散制御方法

(iv) ｂｉｔ−ｍａｐ応答を用いた半結合方法

これらを用いて、以下の処理を基本にして、等結合述語の積から成る問合せは分散処理される。

(1) 結合属性ａについて、あるリレーションＲのａについての射影Ｒ〔ａ〕の放送

(2) ａを有するリレーションを持っているデータモヂュール（ＤＭ）での結合

(3) 結合結果のａについての射影のＲ〔ａ〕に対するｂｉｔ−ｍａｐの放送

(4) 他のＤＭからのｂｉｔ−ｍａｐの積を取ることによる結合処理

これによって、各ＤＭのリレーションの属性ａの値は全て等しくなる。全ての結合属性について、上記の処理を、順々に行う。ある結合属性ａの処理において、以前に処理された結合属性ｂの値が制限される時は、このｂの値のｂｉｔ−ｍａｐも、(3)にあわせて放送される。次にどの結合属性の処理を行い、どのリレーションを放送するかは、(3)の応答に乗せられてくるパフォーマンス情報（各属性のリレーション内でのカーディナリティ）に基づいて、各ＤＭ（ＧＤＰ）で、同時にしかも同一の決定がなされる。この時の各属性ａの通信コストは、(1)での１つの射影Ｒ〔ａ〕の放送と、(3)でのｎコのＤＭからのｂｉｔ−ｍａｐの放送コストである。一方処理コストとしては、ｂｉｔ−ｍａｐ演算は無視し得るとすると、ｎコのＤＭでの結合処理のコストとなる。

現在、我々は、ローカルネットワークを所有していたので、当協会のＪｉｐｎｅｔ〔ＹＡＭＡＫ73 ＩＴＯ　Ｔ７４〕の提供するＩＴＰ（対話プロトコル）を用いて、ローカルネットワークをシュミィレートした通信システム上に、分散型データベースシステムＪＤＤＢＳを実現しようと考えている〔図2.35〕。ＪＤＤＢＳは、次の３つのＤＭから成るとする。

(1) DM_1（ＰＲＤＢS_1）　Ｍ−１７０Ｆ

(2) DM_2（ＰＲＤＢS_2）　Ｍ−１７０Ｆ

(3) DM_3（ＰＲＤＢS_3）　Ｍ−１７０Ｆ

(4) DM_4（ＰＲＤＢS_4）　Ａ−ｃｏｓ７００

ＰＲＤＢＳは、３章で述べる様に、我々がＡＩＭ及びＡＤＢＳ上に実現したプロジェクト管理データベースシステムである。上の３つのＰＲＤＢＳは全て同一のスキーマとオカーランスから成っているとする。各々のＤＭは、３章で示す同一のリレーショナルスキーマが定義されている。これに対して各ＬＣＳは、図2.4の様なビューとして次の様に定義させたものについて考える。

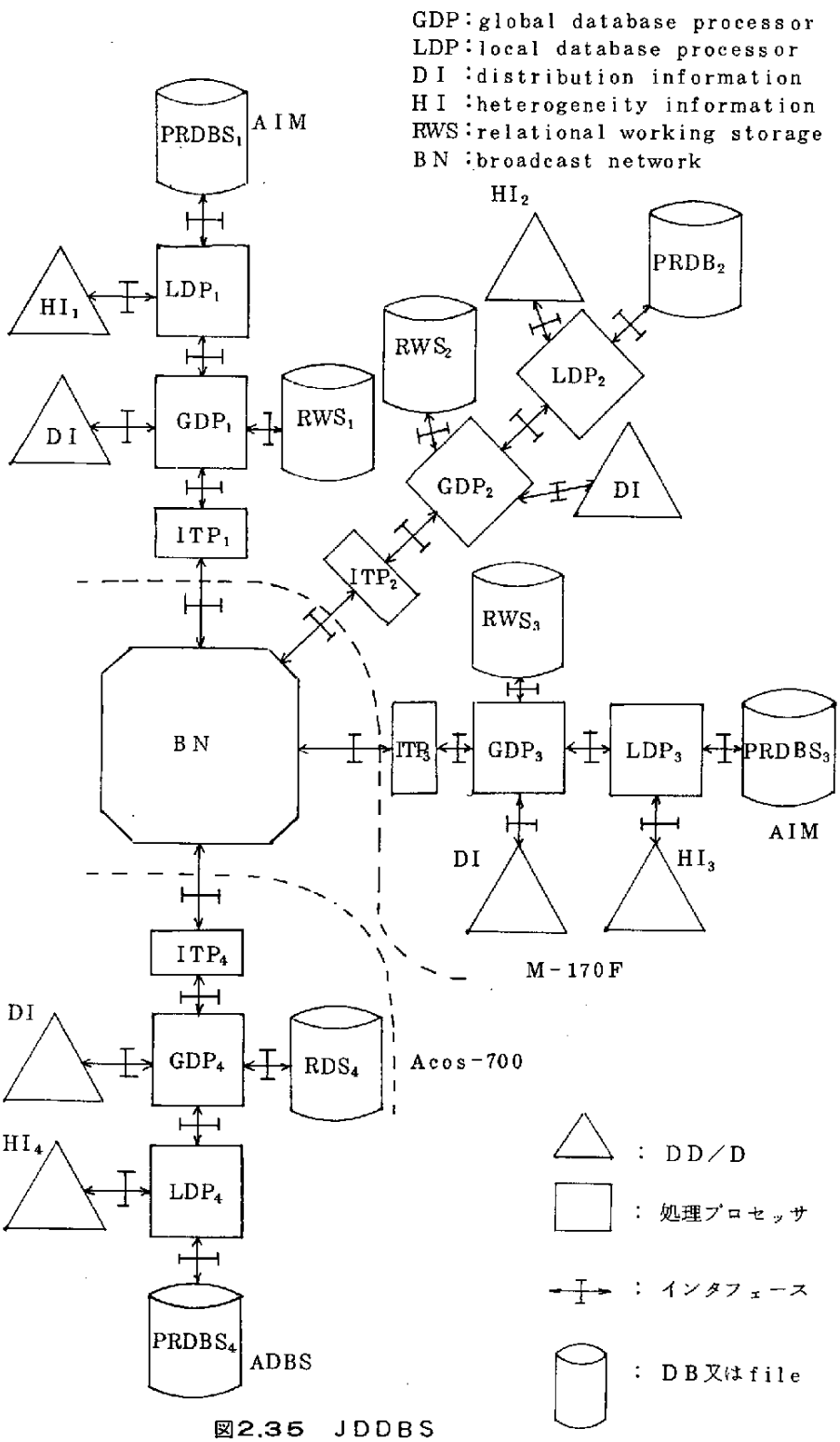

GDP : global database processor
LDP : local database processor
DI : distribution information
HI : heterogeneity information
RWS : relational working storage
BN : broadcast network
PRDBS1
AIM
HI1
LDP1
DI
GDP1
RWS1
ITP1
HI2
PRDB2
RWS2
LDP2
GDP2
DI
ITP2
RWS3
BN
ITP3
GDP3
LDP3
PRDBS3
AIM
DI
HI3
M-170F
ITP4
DI
GDP4
RDS4
Acos-700
HI4
LDP4
PRDBS4
ADBS
: DD/D
: 処理プロセッサ
: インタフェース
: DB又はfile

図2.35 JDDBS

```
DM1 :  range (e,EMPLOYEE) (er,EMP-REP-LNK);
       range (r,REPORTR);
       define PRDBS
            EMP(e.@E,e.enu,e.ename,
                 age=1982-e.birth-year,r.@R)
            where e.@E=er.@E and er.@R=r.@R;
DM2 :  range (j,SUBJECT)(P,PROJECT)(r,REPORTR);
       range (pj,PROJ-SUBJ)(rj,REP-SUBJ);
       range (d,PRODUCT)(pd,PROJ-PROD);
       define PRDBS2
            SUBJ (j.@J,p.@P,r.@R)
            where
                 j.@J=pj.@J and pj.@P=p.@P and
                 j.@J=rj.@J and rj.@R=r.@R:
            PROD(p.@P,d.@D,d.pd-name,d.cost)
            where
                 p.@P=pd.@P and pa.@D=d.@D;
DM3 :  range (r,REPORTR);
       define PRDBS3(r.@R,rname=r.title,
                        pub-year=r.pyear);
DM4 :  range (p,PROJECT)(pe,PROJ-EMP)(e.EMPLOYEE);
       define PRDBS4
            PROJ-EMP(p.@P,p.pname,p.pno,e.@E)
            where
                 p.@P=pe.@P and pe.@E=e.@E;
```

図2.35のBNは、放送機能のシミュレータである。BNと各DMとは、ITPを介して通信を行う。あるDMから出されたメッセージは、他の全てのDMに於いて受信可能である。各DMは、BNからの送信要求によって、常にメッセージの受信を行う。DMは、BNに対してpassiveである。DMは、受けたメッセージが自分と無関係であれば、受信をやめる。受信したリレーションは、各DM_i のRWS_i に格納され、リレーションのスキームはDIに格納される。DMが送信したい時は、CSMA／CD〔METCR76〕に基づいて、送信の競合が制御される。来年度、図2.35の実現を行う予定である。

2.3 分散更新処理

分散型データベースシステム（ＤＤＢＳ）における、更新要求の分散処理について述べる。更新処理に於いては、各ＤＢＳ内、及びＤＢＳ間でのインテグリティ条件が保たれている必要がある。

ＤＤＢＳは、ＧＣＳレベルで、データ項目Ｘ，Ｙ，Ｚ……から成るとする。これらの仮想データＸは、実際には、ＬＣＳレベルのデータx_1，…，x_nに対応するとする。

ここで、x_1，…，x_nは、Ｘのコピーである（$m \geqq 1$）。各データモデュールＤＭ，x_1，…，x_nを格納している。

2.3.1 トランザクション

更新においてトランザクション概念は重要である。トランザクションとは、ユーザにとって、原子的（ａｔｏｍｉｃ）な実行単位である。即ち、トランザクションの実行中の中間結果を他のトランザクションは見ることは出来ない。又、トランザクションの実行は、全て行なわれる（コミットされる）か、全く行われないかのどちらかである。コミットされたトランザクションの結果は、データベースの物理的な状態変化をもたらし、他のユーザによって見られることができる。コミットされないトランザクションの結果は、データベース上に、全く反映されない。

ＧＣＳレベルの以下のＱＵＥＬ基本演算を、ＧＣＳユーザのトアンザクションとする。

(1) append

(2) delete

(3) replace

(4) retrieve

(1)～(3)の更新演算は、更新データを見つける為の検索と、これに続くデータベースへの書き込み（ｗｒｉｔｅ）としてモデル化される。前者の検索に対しては、2.2節で述べた問合せの分散処理よって、最適な通信処理を行い、必要な更新データを結果ＤＭに生成するものとする。この結果ＤＭ内の更新データによって、他のＤＭ内のデータの更新が行われる。

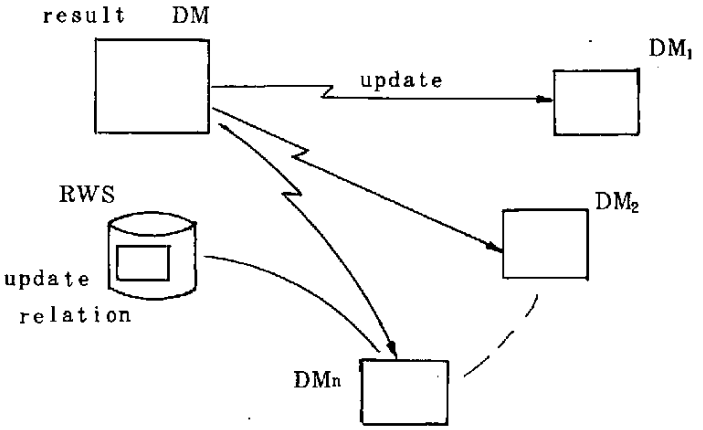


図2.3.6 更新処理

GCSレベルのアクセス要求は、トランザクションとして、以下の基本演算から成っている。

GCS trans	begin−trans	
	end−trans	
	read(x)	xのread
	write(x,val)	xに値valをwriteする

このGCSトランザクションは、LCS上のトランザクションに変換される。この時LCSレベルのデータに対して、各DM上で、次の基本演算が実行される。

dm−read(x)	xのread
dm−write(x)	xのwrite

2.3.2 スケジュールと同期手法

GCS transactionのreadとwriteは各データモデュールDM_1, …, DM_m 上のdm−read dm−write演算として実行される。ここで、以下の記法を導入する。

$\Pi \triangleq \{T_1, \cdots, T_l\} \triangleq$ l コの transaction $T_1, \cdots, T_l$ の集合

$r_i^k[x] \triangleq T_i$ のdm−readで、DM_k 内のxに対して作用する

$w_i^k[x] \triangleq$ 〃 dm−write 〃

$op_i^k[x] \in \{r_i^k[x], w_i^k[x]\}$

ここで、各DM_i に対して、以下のlog_i を定義する。

$log_i \triangleq DM_i$ で実行される演算のシーケンス

この時、Π に対するスケジュールは、$log_1, \cdots, log_m$ の集合となる。

Πのserialスケジュールは、以下の様に定義される。

$T_1, \cdots, T_l$ に、以下の条件を満足する全順序$T_{p_1} < T_{p_2} < \cdots < T_{p_l}$ が存在する。

即ち、全てのlog_k に対して、$T_i < T_j$ ならば、全てのT_i の演算op_i^kは、T_jのどのop_j^kよりも前に実行されるこの$T_{p_1} < \cdots < T_{p_l}$をserial orderと呼ぶ。即ち、どのlogにおいても、各transactionの演算は、inter leaveして行われず、どのlogでも同一の順序で実行される。

$op_i^{ki}(x_i)$ をT_i の, $op_j^{kj}(x_j)$ をT_j の演算とする。この時、

2つの演算が同一のDMで実行され($k_i = k_j$)、同一のデータに作用し($x_i = x_j$)、かつ、

2つのどちらかが少しでもdm−writeの時、op_i^{ki}とop_j^{kj}とはconflictすると呼ぶ

T_1 : $r_1^1(\alpha)$ $r_1^2(\beta)$ $w_1^2(\beta)$ $w_1^3(\gamma)$

T_2 : $r_2^2(\beta)$ $w_2^3(\sigma)$ $r_2^1(\delta)$ $w_2^3(\gamma)$

S_1 {
log_1 : $r_1^1(\alpha)$, $r_2^1(\delta)$
log_2 : $r_1^2(\beta)$, $w_1^2(\beta)$, $r_2^2(\beta)$
log_3 : $w_1^3(\gamma)$, $w_2^3(\sigma)$, $w_2^3(\gamma)$

図2.37 シリアルスケジュール

{
log_1 : $r_2^1(\delta)$, $r_1^1(\alpha)$
log_2 : $r_1^2(\beta)$, $w_1^2(\beta)$, $r_2^2(\beta)$
log_3 : $w_1^3(\gamma)$, $w_2^3(\gamma)$, $w_2^3(\sigma)$

$S_1 = S_2$

図2.38 シリアライザブルスケジュール

Π 合 $\{T_1, \cdots, T_1\}$ とし、Πのあるスケジュールを $E = \{log_1 \cdots, log_m\}$ とする。このスケジュールEが正しいとは、Πから生成されるあるserialスケジュールと等しい時（serializable）時である。Eがserializableとは、以下の条件を満足する時である

即ち、次の様な全順序がΠ内に存在する（$T_{p_1} < T_{p_2} < \cdots < T_{p_1} \cdots\cdots$

$\{P_1, \cdots, P_l\}$ は $\{1, 2, \cdots, l\}$ のあるpermutation）

T_i と T_j 内の全てのconflict演算 op_i と op_j とに対して、

$T_i < T_j$ ならば、op_i と op_j の属する log_k 内で、$op_i \rightarrow op_j$

この $T_{p_1} < T_{p_2} < \cdots < T_{pn}$ serialization orderと呼び、Eの実行結果は、T_{p_1} …, T_{pn} とserialに実行した結果と等しい。

DDBSにおけるconcurrency controlとは、conflictする演算の相対的な実行順序を制御することである。

同期手法（synchronization technique）とは、concurrency control を実行するためのアルゴリズムである。

serialization theory は、次の様にも表わせる〔Bernstein, P.A〕

ここで、T_i , T_j を異った2つのtransactionとする。

$T_i \rightarrow_{wr} T_j$ ＝どのlogでも、あるデータxに対してT_jは、T_iがreadした後にwriteする。

$T_i \rightarrow_{rw} T_j$ ＝どのlogでも、あるデータxに対して、T_jは、T_iがwriteした後にreadする。

$T_i \rightarrow_{ww} T_j$ ＝どのlogでも、xに対して、T_iはT_jがwriteした後にwriteする。

$T_i \rightarrow_{rwr} T_j = T_i \rightarrow_{rw} T_j$ 又は $T_i \rightarrow_{wr} T_j$

$T_i \rightarrow T_j = T_i \rightarrow_{rwr} T_j$ 又は $T_i \rightarrow_{ww} T_j$

この時、スケジュールEは、

(a) $\rightarrow_{rwr}$と$\rightarrow_{ww}$関係は、acyclicでかつ

(b) 全ての$\rightarrow_{rwr}$と$\rightarrow_{ww}$とconsistentな全順序$T_{p_1} < T_{p_2} < \cdots < T_{p_i}$が存在する時、serializableである。

これは、同期手法として、

(i) r－w conflict (→rwr)

(ii) w－w conflict (→wu)

とを、

(iii) transの全順序を保つ限り

独立して、同期させることができることを示している。(i)、(ii)の各々に対する同期手法を、

r－w synchronization

w－w synchronization と呼ぶ。

同期手法としては、次の2種がある〔BERNP80C〕

(i) two－phase locking (2PL)

(ii) time－stamp ordering(T／O)

2.3.3 2フエーズロツク(2PL)

locking ruleは次の様である。

(a) readする前にRlock

(b) writeする前にWlock

(c) T_iとT_j ($T_i \neq T_j$)とは、同時にconflicting lockを持てない

(d) Tは、xがTのlockを既に持っている時、付加的なlockを行えない

(e) two－phase locking protocolに

即ち、(i) unlock operationが実行された後、lockを行えない

(ii) object への lockは、transの終了時には、全てreleaseされている

lockがconflictするとは、以下のことである。

即ち、O_iとO_jを2つの演算とし、O_i－lock, O_j－lockを各々対応したlockとすると、O_i－lockとO_j－lockとがconflictするとは、O_i〔x〕とO_j〔x〕との実行順序をあるxに対して変えれないことである。R－lockとW－lockとでは、以下の時conflictを

生ずる。 (i) r—w synchronization R—lock(x)とW—lock(x)はconflict

(ii) w—w synchronization W—lock(x)とW—lock(x)はconfliet

A．実現方法

２ＰＬの実現方法としては、以下のものがある。

(i) 基本２ＰＬ

(ii) 主コピー２ＰＬ

(iii) voting ２ＰＬ

(iv) centralized ２ＰＬ

これらは、データオブジェクトXが、コピーをＬＣＳ上に（x_1,…,x_n）もつ時のlockのかけ方の差によって区別される。

(1) 基本２ＰＬ

全てのコピーに対して、Rlock,Wlockかけられる。

(2) 主コピー２ＰＬ

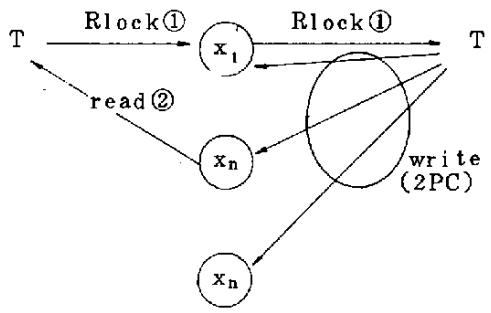
コピー｛x_1,…,x_n｝内の１つが主コピー（x_1とする）と呼ばれる。

readは、主コピーx_1 Rlockし、x_iをreadする。

writeは、x_1のがWlockされ全てのx_1,…,x_1をwritする。

read時に、x_1をRlockするための通信overheadがある。

write時には、x_1だけをwlockするだけですむので通信overheadが少い。



(3) voting ２ＰＬ（多数決法）

(i) ww—synchronization

全てのコピーx_1,…,x_mに、Wlockを出す。

この内、過半数がWlockを行えたならば、writeを行う。

(ii) rw—syhchronization

readを行う時、全てのx_1,…,x_nにRlockが必要 ⇒ more communication

(4) centralized ２ＰＬ

２ＰＬのスケジューラは、１カ所に存在し、これが集中制御を行う。

lockは、このスケジューラ内にsetされる。

B．deadlock

locking手法では、deadlockが生じ得る。deadlock問題の解としては、

(i) detection

(ii) prevention の２つの方法がある。

a．deadlock detection

deadlockの検出は、transactionのwait−forグラフがサイクルを持つ時になされる。wait−for graphは、ノードをtransactionとする有向グラフである。有向エッヂ$T_i \rightarrow T_j$は、T_iがT_jによって現在所有されているlockのreleaseを待っていることを示している。DDBSでは、各DMで生成されたwait−for graphから、DM間のwait−for edgeを加えた全体的なwait−for graphを生成せねばならない。deadlockが検出されると、cycle内のあるtransactionがback outされた後に、restartされる。このvictim としてのtransactionは、backout−restartのコストが最少となる様に決定される。

b．deadlock prevention

deadlock prevention方法は、deadlockが起りそうな時に、あるtransactionをabort(back out)するものである。

Thomas〔THOMR79〕による方法では、トランザクションに、あらかじめタイムスタンプを与えておく。今トランザクションT_iが、現在T_jが使用しているデータを必要とする時、($T_i \rightarrow T_j$と記す)、トランザクションのタイムスタンプの値によって、一方をbackoutしてしまうことによって、デッドロックの可能性を除いてしまう方法である。

この方法として、次の2つがある。

(i) wait−die方式

(ii) wound−wait方式

wait−die方式では、T_iがT_jの使用中のデータを要求した時、T_iのタイムスタンプ値(TM(T_i)と記す)が、T_jのものより大きい(T_iの方がT_jより若い)時、T_iをbaek outし、同一のタイムスタンプ値で再スタートさせる。そうでなければ(T_iの方が古ければ)そのままT_iにT_jのデータの解放を待たせる。

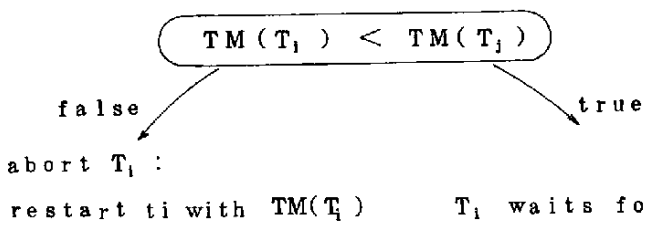


図2.37 wait−die方式

wound−wait方式では、T_iがT_jより若ければ、T_jをabortし、再スタートさせる。そうでなければ、T_iにT_jのデータ解放を待たせる。

デッドロック防止では、トランザクションの繰り返し再スタート(cyclic restart)を防止することがパフォーマンス上重要である。wound−wait方式では古いトランザクションが数多く再スタートされ得る。この為、wait−die方式よりもより少い再スタートをも

たらす。

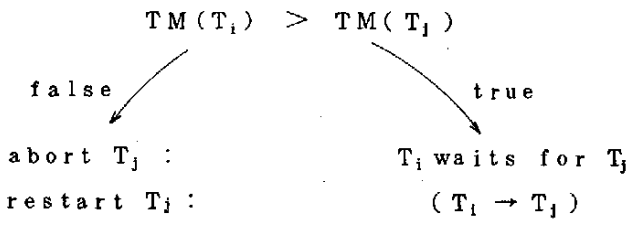


図2.40 wound-wait方式

2.3.4 タイムスタンプ順序づけ（T／O）

timestamp ordering(T/O)方法は、serialization orderを前もって定め、transactionは、この順に実行させる方法である。

DM上での各演算(e.g. dm-read, dm-write)は、transactionを入力したtransaction module(TM)によってtimestampが与えられる。各DMは、conflictするoperationを、このtimestampの順序に基づいて実行する。

conflict operationは、以下の様である。

(i) rw-conflict dm-write$_i$(x)とdm-read$_j$(x)(i≠j)

(ii) ww-conflict dm-write$_i$(x)とdm-write$_j$(x)(i≠j)

T／Oの実現方法としては、以下の4つがある。

(i) basic T/O

(ii) thomas write rule(TWR)

(iii) multi-version T／O

(iv) conservative T／O

各々には、two-phase commitを行うものと、行なわないものとの2つがある。ここでは、後者についてのべる。前者では、pre-commit及びcommitをbufferすることによって処理できる。

A. Basic T／O

各データオブジェクトxは、以下のR-ts(x)とW-ts(x)とを有している。

R-ts(x)≙xに対する過去のdm-read(x)の中で、最も大きなtimestampの値

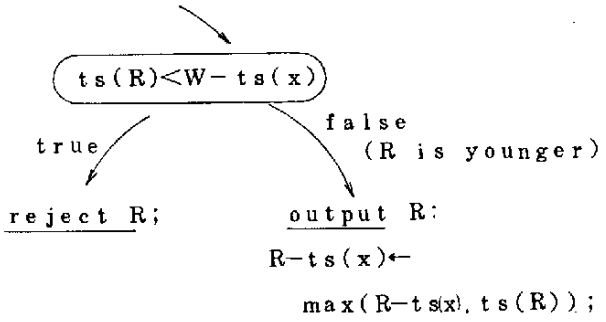
W-ts(x)≙ 〃 dm-write(x) 〃

T／Oスケジューラは、dm-readとdm-writeをTMから受けとり、T／Oに基づいたoperationのsequenceを出力する。（logの出力）

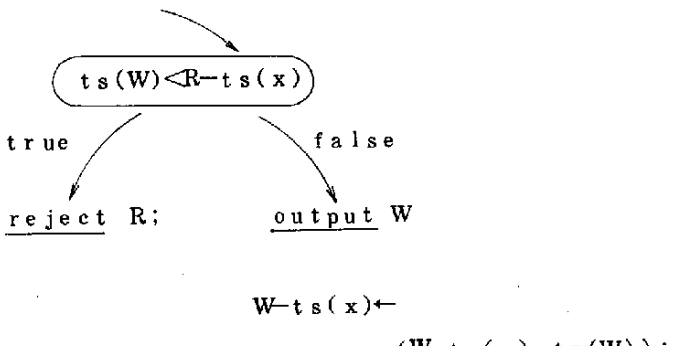
a．no two-phase commit

(i) rw-synchronization

R：dm-read(x) with ts(R)

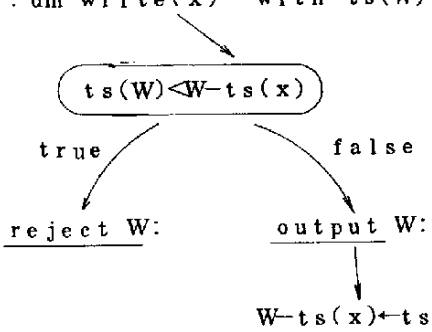


W：dm-write(x) with ts (W)



(ii) ww-synchronization

W：dm-write(x) with ts(W)



rejectされたoperationは、TMでabortされ、新たなtimestampをもらって、restartされる。

b．two-phase commit T／O

two-phaseコミットを取り込んだ方式である。ここでは、各DMのT／Oスケジューラは、precommit(x)を受けいれ、バッファされる。スケジューラは、タイムスタンプTSを持ったprecommit(x)を受信したならば、このprecommitのwriteが出力されるまで、TSより大きなタイムスタンプを持つ他のトランザクションのwriteを出

力してはならない。

この為、以下のレジスタが、各データオブジェクトx毎に必要となる。

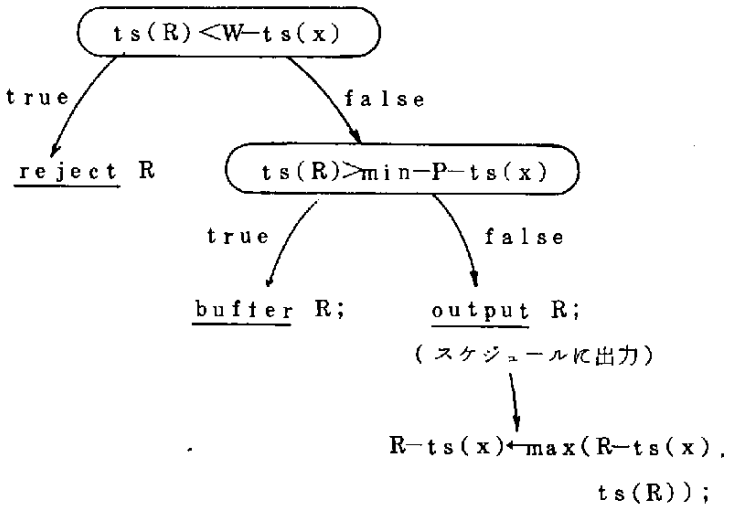
min−R−ts(x)≙ バッファ内のread(x)の最少のタイムスタンプ

min−W−ts(x)≙ バッファ内のwrite(x)の最少のタイムスタンプ

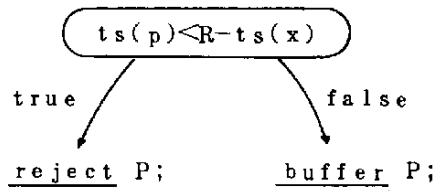
min−P−ts(x)≙ バッファ内のprecommicの最少のタイムスタンプ

r−w同期手法は、各DBSのT／Oスケジューラが以下の処理を行うことによって実現できる。

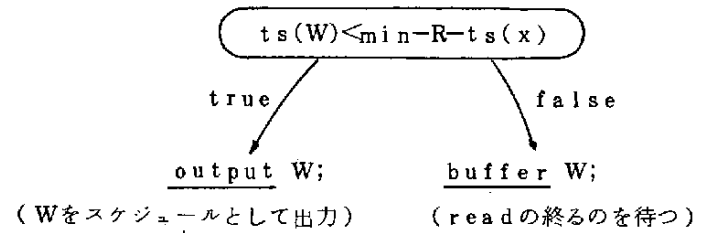
Rに対して(read(x))



P (precommit(x))に対して



W(write(x))に対して、



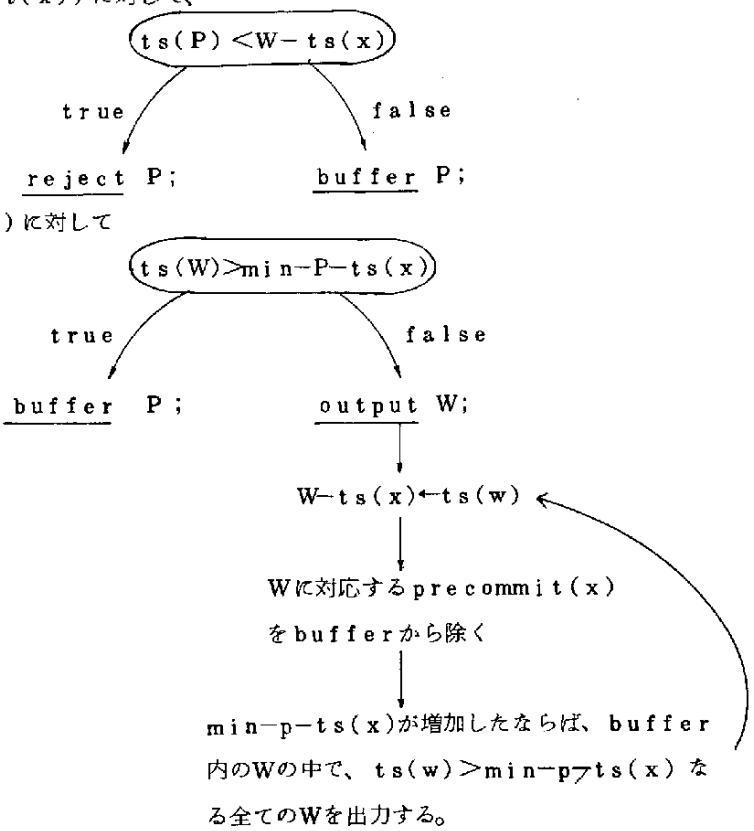
xのprecommit(x)をbufferが除く

min−P−ts(x)が小さくなったならば、

対応するRとWの出力を行う

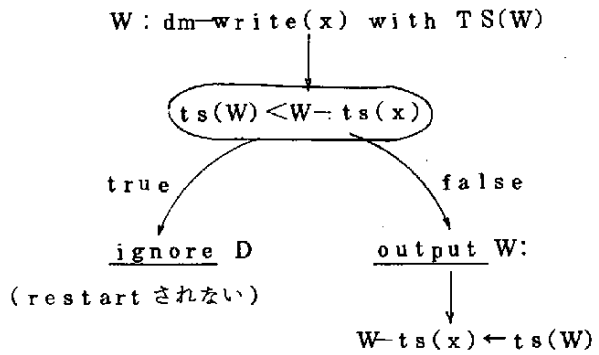
ｗｗ－同期手法は、以下の様である。

Ｐ（ｐｒｅｃｏｍｍｉｔ(ｘ)）に対して、



Ｂ．ＴＷＲ（Ｔｈｏｍａｓ ｗｒｉｔｅ ｒｕｌｅ）

ｗｗ－ｓｙｎｃｈｒｏｎｉｚａｔｉｏｎに用いられるＴＷＲでは、ｗｒｉｔｅはｒｅｊｅｃｔされずに、無視される。



TWRは、常にprecommitを受けつけ、dm−writeをbufferしない。この為、 2 phaseコミットを取り込む必要はない。

C．multi−version T／O

各データobjectは、dm−read(x)とdm−write(x)のhistoryを有している。ここで、

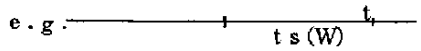
R(x)≙ x上に今まで実行されたdm−read(x)のtimestampの集合

W(x)≙ x上に今まで実行されたdm−write(x)のtimestampと、その値<W−ts val>(version)の集合

R ≙ dm−read(x)

W ≙ dm−write(x)

interval(W)≙ W(x)内で、ts(W)よりも大きくてかつ最少のW−tsを有するversionとts(W)とのinterval



interval(W)=(ts(W),t)

a．no two−phase commit

(i) rw−synchronization

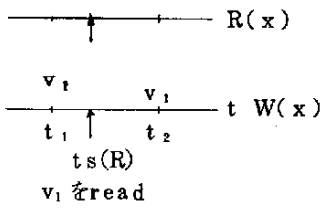
R：dm−read(x) with ts(R)

↓

W(x)の中で、ts(R)よりも小さく最大のW−tsを持つversionの値をreadする

↓

R(x)に、ts(R)を加える。



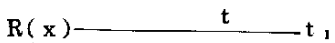
W: dm−write(x) with ts(w)

↓

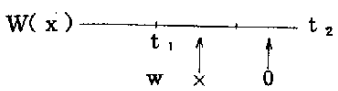
interval(W)内に、R(x)内のP−tsが存在すれば

true → reject W:

false → output W:



W(x)に(ts(W),val)を加える



(ii) ww− synchronization

W

W(x) にnew version < ts(w)|val> を加える

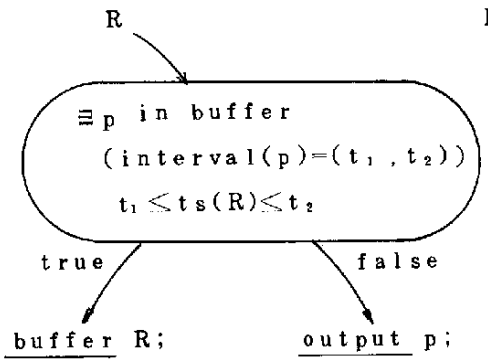
b．two−phase commit

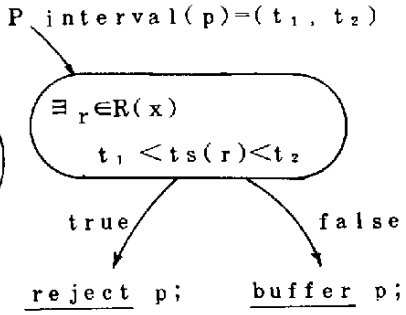
dm−readとdm−writeとは、bufferされる。この為以下の記法を導入する。

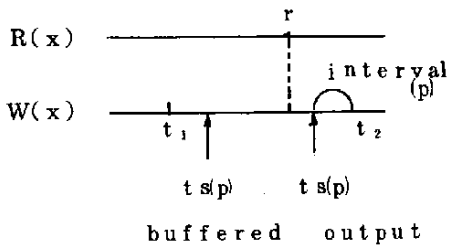
P≙precommit(x)

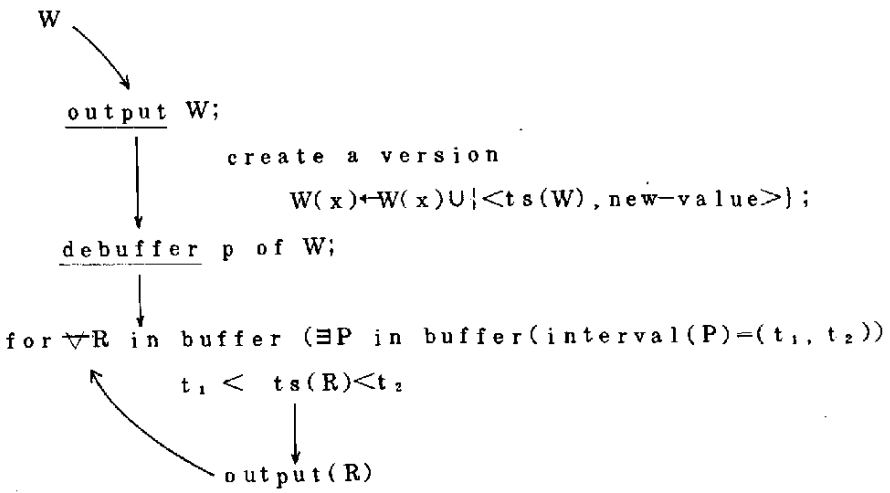
interval(p)≙ts(W)から W−ts(x) > ts(p)なる最少のW−ts(x)までのインタバル

(i) rw−synchronization



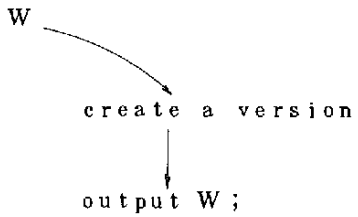






(ii) ww−synchronization

dm−writeは、rejectされない。dm−writeに対しては、new versionが生成される。



D. Conservative T／O

各DMのT／Oスケジューラsは、各TM_iから、演算を、timestamp順に受けとるとする。（networkは、FIFO channelと考えられる）

DMで、operationを実行する為には、全てのTM_iで、このoperation より以前の、operationが到着し、処理された後にこのoperationが処理される。これによって、transaction の restartを防止する。

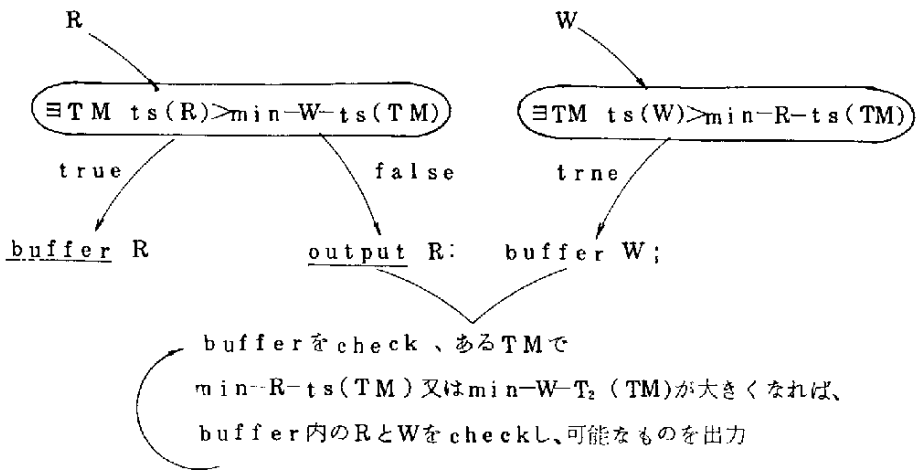
ここで、

TM_i ≙ transaction module

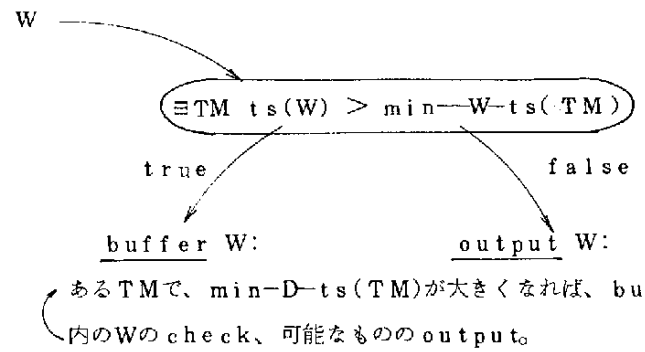
min−R−ts(TM_i) ≙ TM_i からのdm−readで、bufferされたものの中で、最も小さいtimessamp値

min−W−ts(TM_i) ≙ 〃 dm−write 〃

(i) rw−synchronization



(ii) ww−synchronization



conservative T／O 方式は、以下の問題点を有している。

(i) あるトランザクションモデュールが、各DMのT／Oスケデューラに何の演算も転送しないと、スケジューラは、これが届くまでwaitしてしまう。

(ii) 競合していないdm−write演算も、全てタイムスタンプ順に処理してしまう。この欠点を解決する為に、SDD−1〔BERNP80a、b〕では、次の様な手法を用いている。

(a) (i)を解決する為に、あるDMに対して転送する演算が存在したい時、タイムスタンプ値のみを運ぶnull演算を送る。方法としては、DMが、あるトランザクションモデュールからある期間、何の演算も送られてこない時、問合せを行いnull演算を求める。以降、何の演算も、あるDMに送ることのないトランザクションモデュールは、無限大のタイムスタンプをつけてnull演算を送る。しかし、巨大ネットワークでは、これらの処理の為の通信オーバヘッドが問題となる。

(b) トランザクションクラスを設ける。定型業務では、各トランザクションの入力とするデータ集合（read−set）と、出力されるデータの集合（write−set）とは、前もって解っている。トランザクションを、これらのwrite−set，read−setによってクラス分けする。トランザクションTのread−set(rs(T))が、あるトランザクションクラス(C)のread−set(rs(C))，wvite−set(ws(C))に対して、

$$ws(T) \subseteq ws(C)$$

$$rs(T) \subseteq rs(C)$$

が成り立つならば、Tは、クラスCに属するという。各クラスは、専用のGDPを有している。Tは、クラスC用のGDPに入力される。

トランザクションを上記した様に分類することによって、トランザクションの競合関係をあらかじめ知る事が出来る。従って、次の様に、タイムスタンプ値を較べる為に、待つ必要のあるトランザクションモデュールの範囲を限定できる。

(i) rw−synchronization

dm−read(x)(=R)をDMが処理する為には、ts(R)より小さな全てのdm−write(x)を待つ必要がない。xをwrite−setに持つ他のクラスのトランザクションか

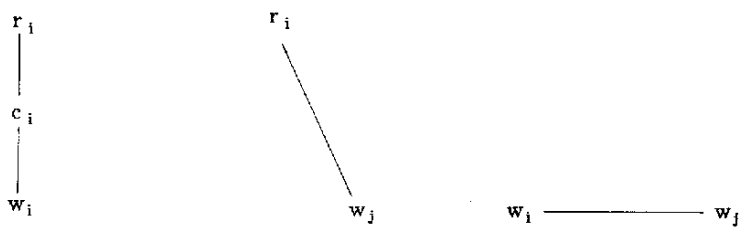
らのdm−write(x)を待てばよい。dm−write(x)(=W)を、DMが処理する為には、xをread−setとして持つ他のクラスのトランザクションからのdm−read(x)をただ待てばよい。

(ii) ww−synchronization

dm−write(x)を処理する為には、このxをwrite−setとしてもつクラスのトランザクションからのdm−writeだけを待てばよい。

(c) 競合グラフ(conflict graph)

種々のトランザクションからの競合演算が、タイムスタンプ順に実行されねばならない場合を限定する為に、次の様な競合グラフを用いる。競合グラフは、異ったクラス内のトランザクション間の可能な競合を表している。競合グラフは、クラス(C_i)，そのwrite−set(w_i)，read−set(r_i)を表すノードと、図2.41のノード間の3種の辺からなる。



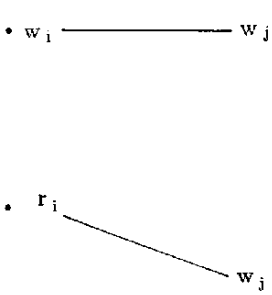
r_i =read−set(C_i)	($i \neq j$)	($i \neq j$)
w_i =write−set(C_i)	$r_i \cap w_j \neq \phi$	$w_i \cap w_j \neq \phi$
(1) vertical edge	(2) diagonal edge	(3) horizontal edge

図2・41 競合グラフ

水平辺は、クラスC_iとC_jのwrite−setが同一のデータを持ち(write−set(C_i)∩write−set(C_j)≠φ)っている場合である。この時DMがC_iからのdm−write(x)(=W_i)を処理する為には、C_jからの全てのdm−write(x)(=W_j)(ts(W_j)<ts(W_i))が到着するのを待たねばならない。

垂直辺は、read−set(C_i)∩write−set(C_j)≠φを表す。この時C_iのdm−read(x)(=R_i)を処理する為には、C_jからのts(R_i)より小さいタイムスタンプを有した全てのdm−write(x)(=W_j)を待たねばならない。逆に、C_jのdm−write(x)(=W_j)を処理する為には、ts(W_j)より小さなタイムスタンプを有したC_iからの全て、dm−read(x)を待たねばならない。

この時、あるDMでC_iからの演算op_iと、C_jからの演算op_jとが、タイムスタンプ順に処理されねばならないのは、次の場合になる。



(α) 辺（w_i，w_j）が、競合グラフ内のあるサイクルの中にあるか

又は

(β) $w_i \cap w_j$ 内のある２つのデータｘとｙが異ったＤＭに存在している

(α) 辺（r_i，w_j）がサイクル内にあるか

又は

(β) $r_i \cap w_j$ 内のある２つのデータｘとｙが異ったＤＭに存在している

ＳＤＤ－１では、クラスをあらかじめ定義しておき、競合グラフを解析することによって、トランザクションの処理を行う為に必要な同期レベルを定めている。これは、定型的な業務で、トランザクションクラスを、あらかじめ定義出来る場合には、有効である。

Ｅ．タイムスタンプの管理

Ｔ／Ｏ同期方式の問題は、データに対するタイムスタンプを格納する事のオーバヘッドである。タイムスタンプは、次の様なＲ－ｔａｂｌｅとＷ－ｔａｂｌｅとによって管理される。

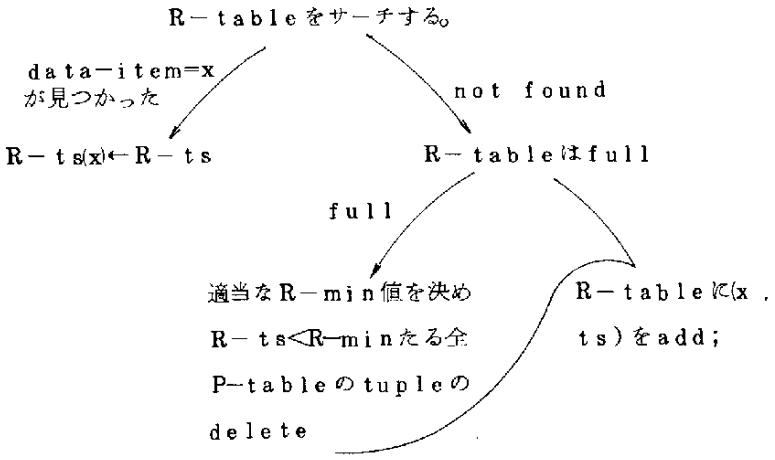
Ｒ－ｔａｂｌｅ（ｄａｔａ－ｉｔｅｍ，Ｒ－ｔｓ）

Ｗ－ｔａｂｌｅ（ｄａｔａ－ｉｔｅｍ，Ｗ－ｔｓ）

ここで、Ｒ－ｔｓとＷ－ｔｓは、データ項目（オブジェクト）ｘに対する各々ｒｅａｄとｗｒｉｔｅに対するタイムスタンプである。ここで、

Ｒ－ｍｉｎ≙ テーブルから除かれたタイムスタンプの中の最大値

Ｒ－ｔｓ(ｘ)を見つける為には、次の様にする



Ｗ－ｔｓ(ｘ)は、Ｗ－ｔａｂｌｅをサーチして見つける。

Ｃｏｎｓｅｒｖａｔｉｖｅ Ｔ／Ｏでは、各データオブジェクト毎のタイムスタンプは不要である。ただし、各ＤＭは、これをアクセスするトランザクションクラスが全て定っていなければならない。

2.3.5　２ＰＬとＴ／Ｏの混合

Method	rw technique	ww technique
1	Basic T/O	Basic T/O
2	Basic T/O	Thomas Write Rule (TWR)
3	Basic T/O	Multiversion T/O
4	Basic T/O	Conservative T/O
5	Multiversion T/O	Basic T/O
6	Multiversion T/O	TWR
7	Multiversion T/O	Multiversion T/O
8	Multiversion T/O	Conservative T/O
9	Conservative T/O	Basic T/O
10	Conservative T/O	TWR
11	Conservative T/O	Multiversion T/O
12	Conservative T/O	Conservative T/O

ＳＤＤ－１

Method	rw technique	ww technique
1	Basic 2PL	Basic 2PL
2	Basic 2PL	Primary copy 2PL
3	Basic 2PL	Voting 2PL
4	Basic 2PL	Centralized 2PL
5	Primary copy 2PL	Basic 2PL
6	Primary copy 2PL	Primary copy 2PL
7	Primary copy 2PL	Voting 2PL
8	Primary copy 2PL	Centralized 2PL
9	Centralized 2PL	Basic 2PL
10	Centralized 2PL	Primary copy 2PL
11	Centralized 2PL	Voting 2PL
12	Centralized 2PL	Centralized 2PL

ＩＮＧＲＥＳ

Method	rw technique	ww technique
1	Basic 2PL	Basic T/O
2	Basic 2PL	TWR
3	Basic 2PL	Multiversion T/O
4	Basic 2PL	Conservative T/O
5	Primary copy 2PL	Basic T/O
6	Primary copy 2PL	TWR
7	Primary copy 2PL	Multiversion T/O
8	Primary copy 2PL	Conservative T/O
9	Centralized 2PL	Basic T/O
10	Centralized 2PL	TWR
11	Centralized 2PL	Multiversion T/O
12	Centralized 2PL	Conservative T/O

２ＰＬとＴ／Ｏのｍｉｘ

２ＰＬにおけるｌｏｃｋｅｄｐｏｉｎｔ（ｇｒｏｗｉｎｇ ｐｈａｓｅの終り）に、ｔｉｍｅｓｔａｍｐを与える。

Method	rw technique	ww technique
13	Basic T/O	Basic 2PL
14	Basic T/O	Primary copy 2PL
15	Basic T/O	Voting 2PL
16	Basic T/O	Centralized 2PL
17	Multiversion T/O	Basic 2PL
18	Multiversion T/O	Primary copy 2PL
19	Multiversion T/O	Voting 2PL
20	Multiversion T/O	Centralized 2PL
21	Conservative T/O	Basic 2PL
22	Conservative T/O	Primary copy 2PL
23	Conservative T/O	Voting 2PL
24	Conservative T/O	Centralized 2PL

図2.4.2　２ＰＬとＴ／Ｏの混合

同期方式は、2.3.2で述べた様に、read—write(rw), write—write(ww)の同期方式を、トランザクション間の全順序関係を保ちさえすれば、別々にすることができる〔BERNP80〕。可能な組合せとしては、図2.42〔BERNP80〕に示す様に48通りある。今まで提案されてきた同期方式は、これらの組合せの中のどれかである。今後は、個々の方法パフォーマンスを明らかにする事が必要となる。

2.3.6 まとめと今後の課題

本節では、分散型データベースシステム(DDBS)に於ける更新要求の分散処理方式の現状について述べた。トランザクションの原子性を保障する為には、2フェーズコミット(2phase—commit)方法を用いる。トランザクションに対して、あるGDP(例えばトランザクションの入力DM)に、分散問合せ処理によって更新データ(更新するデータと更新されるデータ)を生成し、その後に更新すべきデータを保有しているDMに更新を行う。この更新は、2フェーズコミット方法によって、更新の原子性を保障する。1対N通信機能を通信ネットワークが有していれば、更新データを放送することによって、関連するDMに、更新データ(及び命令)を同時に届ける事が出来る。

各DMで、更新演算(readとwrite)は、応答性を高める為に、各演算のI/O時間を用いて、インタリーヴして実行される。この時、競合する演算にとって、その実行順序の違いは、異った結果をもたらしてしまう。この為、各DMで、2つのトランザクションT_iとT_jの競合する演算を全て同一の相対的順序で実行させねばならない。この制御技法を同期方式と呼ぶ。
同期方式と呼ぶ。同期方式としては、次の2種が考えられる。

(i) 2PL(2phase locking)

(ii) T/O(time—samp ordering)

又、同期方式は、readとwrite(rw—synchronization), writeとwrite(ww—synchronization)との同期を、トランザクション間のある全順序関係を保ちさえすれば独立な方法を用いることができる。この組合せとしては、図2.42の様に48通りの可能性がある。今後、これらの方式の中から、具体的な運用環境の中で最適な方式を選択する必要がある。

DDBSに於ける更新処理に於ける他の問題としては、以下のものがある

(i) GCS更新要求から全体LCS更新要求への表現変換

ビュー更新と同様な問題が生じる。GCSリレーションをLCSリレーション上の抽象データ型と考えるのが一つの解である。

(ii) 異種データベースシステムの同時更新

LCSレベルに於ける演算単位(e.g.ロックの単位、タイムスタンプの対象)と、各DBSでサポートする演算単位との整合性が必要になる。

(iii) 障害管理

DM及びDDBSの分割障害に対する障害管理方式

2.4 応　　用

異種分散型データベースシステムＪＤＤＢＳの応用としては、以下のものを考えている。

(i) オフィス情報システム

(ii) 意思決定システム

(iii) ＣＡＤ／ＣＡＭ

2.5 まとめと今後の課題

本章では、分散型データベースシステム（ＪＤＤＢＳ）に於ける全体アーキテクチャ、問合せの分散処理方式、更新の分散処理方式について論じた。全体アーキテクチャとしては、異種データベースシステムを統合する為に必要なスキーマ構成（四層スキーマ構造）、これに基づいた具体的なシステムアーキテクチャを示した。我々の目指すＪＤＤＢＳは以下の特徴を有している。

(i) ローカルネットワークの利用（１：ｎ通信）

(ii) 異種データベースシステム（e.g. ＣＯＤＡＳＹＬ　ＤＢＳ）の組み込み

(iii) 個人用データベースシステム

(iv) 更新処理

分散問合せ処理では、ローカルネットワークの放送通信機能を用いたアルゴリズムと、その実現イメージを示した。分散更新処理では、現在の更新技術を、同時更新制御を中心として、現状のサーベイを行った。１：ｎ通信機能は、コミットメントの制御に対して有効と考えられる。

3．CODASYLデータベースシステムのリレーショナル インタフエース(LDP−V1.5)

3・1　序

ＣＯＤＡＳＹＬデータベースシステムは、論理と物理構造が未分離な網型データ構造と、手続的な操作言語とを提供し、主にパフォーマンス指向の大規模定型業務に用いられて来た。しかし、近年データベースシステム応用の多様化と、非定型利用の増大によって、ＣＯＤＡＳＹＬデータベースシステム上の非手続的インタフェースの必要性が高まっている。

分散型データベースシステムにおいても、複数の異種データベースシステムを統合的に利用する為には、異種性問題と分散問題を解決する必要がある〔ＴＡＫＩＭ７８，７９〕。異種性としては、以下の２点がある。

１）データ構造（スキーマ）の相違

２）アクセス言語の相違

異種性問題を解決する為には、同種化、即ち共通モデルに基づいたデータ記述とアクセス言語設定が必要となる。我々は、共通モデルとしてリレーショナルモデルを用いる。

当協会で開発したローカルデータベースプロセッサ（ＬＤＰ−Ｖ1.5）は、ＣＯＤＡＳＹＬデータベースシステムに対して、リレーショナルモデルに基づいたスキーマとアクセス言語(ＱＵＥＬ)によるアクセスをサポートするシステムである。ＬＤＰ−Ｖ1.5の目標は、以下の２点である。

１）現在開発中の分散型データベースシステム（ＪＤＤＢＳ）〔ｃｈ.２〕における異種データベースシステムとしてのＣＯＤＡＳＹＬデータベースシステム上の共通インタフェースとなる。

２）ＣＯＤＡＳＹＬデータベースシステムの一般ユーザインタフェース及びにＣＯＢＯＬ　ＤＭＬプログラムの開発支援ツールとなる。

ＣＯＤＡＳＹＬデータベースシステム上のリレーショナルインタフェースの実現の為には、ＣＯＤＡＳＹＬスキーマからリレーショナルスキーマへの変換と、逆にリレーショナル問合せ（ＱＵＥＬ問合せ）をＣＯＢＯＬ　ＤＭＬに変換し、実行させる問合せ変換〔ＴＡＫＩＭ８０〕が必要である。ＬＤＰ−Ｖ1.5では、検索利用に限定して上記の問題を解決した。更新利用に関しては、第４章の更新非手続的インタフェースＬＤＰ−Ｖ２で論じる。

ＬＤＰ−Ｖ1.5として実現された機能は次の通りである。

１）ＱＵＥＬ検索からＣＯＢＯＬ　ＤＭＬの生成

２）ＣＯＤＡＳＹＬデータベースシステム（ＡＩＭ、ＡＤＢＳ）での実働

３）結合として、主属性間の等価結合、不等価結合の処理

４）非主属性間の結合処理

５）ＬＣＳに対する階層的なビュー定義とビュー問合せ処理

６）ＣＯＤＡＳＹＬスキーマからリレーショナルスキーマの自動生成機能

7）ローカル通信処理（プロセス間通信）

ＬＤＰ－Ｖ1.5は、当協会のＡＩＭ（Ｍ－１７０Ｆ）、ＡＤＢＳ（ＡＣＯＳ－７００）上に実現され、性能面と利用面の評価を行なった。評価から、ＬＤＰ－Ｖ1.5は非定型業務に対して充分に適用可能であることがわかった。

3.2節ではスキーマ変換を、3.3節では問合せ変換を、3.4節ではＬＤＰ－Ｖ1.5の実現を、3.5節ではＬＤＰ－Ｖ1.5の評価を論じる。

3.2 スキーマ変換

ＣＯＤＡＳＹＬスキーマからリレーショナルスキーマ（これをローカル概念スキーマ（ＬＣＳ）〔ch.2, ＴＡＫＩＭ７８, ７９〕と呼ぶ）への変換について論じる。ここでは、検索に限定したスキーマ変換を、更新については第４章で論じる。まず最初に、ＣＯＤＡＳＹＬモデルの定義を行ない、次にリレーショナルスキーマへの変換アルゴリズムについて論じる。

3.2.1 CODASYLモデル

ＣＯＤＡＳＹＬモデルは、データの構造を定義するＣＯＤＡＳＹＬスキーマと、データをアクセスする為のアクセス言語（ＤＭＬ言語）とから成る。

Ａ．ＣＯＤＡＳＹＬスキーマ

ＣＯＤＡＳＹＬスキーマは、データ構造を定義するレコード型とセット型の集合から成る。レコード型、セット型はＣＯＤＡＳＹＬデータベースを記述するものである。これに対し、ＣＯＤＡＳＹＬデータベースの中味を表わすのがレコードオカーランスとセットオカーランスの集合である。

ａ）レコード型とレコードオカーランス

(1) 論理構造

レコード型の外延、レコードオカーランス集合はデータベース内の事象の集合とし、D_0、…D_n を値の集合とする。各D_i は必ずしも互いに異なる必要はない。この時、レコードオカーランス集合Ｒは、次の様に定義される。

$$R \subseteq D_0 \times D_1 \times \cdots \times D_n$$

Ｒの要素ｒは、レコードオカーランスと呼ばれ、次の様に表わされる。

$$r = (d_0, d_1, \cdots, d_n) \quad d_0 \in D_0,\ d_1 \in D_1,\ \cdots,\ d_n \in D_n$$

レコードオカーランスｒ内の各値d_iの意味が、ｒ内のその位置に依存している制約を除く為、各D_i に対して、データ項目ｔｉ を導入する。

$$r = (t_0 / d_0, \cdots, t_i / d_i, \cdots, t_n / d_n)$$

この時、D_i をデータ項目ｔｉの定義域と呼び、$dom(t_i) = D_i$ と表わす。ここで $r[t_i] = d_i$ とする。繰り返し句（repeating qroup）については、繰り返しの各々要素を、独立したデータ項目として考えることにする。従って、繰り返し句は、明には考えないとする。

Rの定義域の中で、Doを各レコードオカーランスを一意に識別する値（これを、データベースキー又は、dbキーの値と呼ぶ）から成る定義域とする。Doは、他のどのレコードオカーランス集合に対しても共通な定義域である。Rの各レコードオカーランスrの中で、ユーザがデータ項目の値として、視ることのできるのはto以外の項目である。

CODASYLモデルの特徴は、レコードオカーランス集合R内のレコードオカーランスrを、一般にto以外のデータ項目によって識別できない（not self-identificable）事である。原則的に、レコードオカーランスrは、R内での位置、即ち、toの値（dbキーの値）によってのみ識別できる。r〔to〕を、db-key(r)と表す。

レコード型Rは、データ項目集合item(R)から成る。レコード型のスキームは、次の様に表わされる。

$$\underline{R}(t_0、t_1、\cdots、t_n)$$

(2) 物理構造

レコードオカーランス集合Rの物理構造としては、レコードオカーランスの格納配置を定める配置モード（location mode）がある。配置モードには、次の4種がある。

（i） CALC

（ii） VIA SET

（iii） SYSTEM

（iv） DIRECT

CALCモードは、あるレコード型Rのあるデータ項目（又は、その集合）の値のハッシュ関数値によって格納するモードである。この時、このデータ項目をCALC項目と呼び、CALC-item(R)と記す。

VIA SETモードは、セット関係を介してレコードオカーランスを格納するモードである。また、SYSTEMモードは、レコードオカーランスの格納をシステムが制御するモードである。DIRECTモードは、格納する位置を直接指定するモードである。

レコードオカーランス集合を物理的に格納する領域の単位はエリア（Area）と呼ばれ、物理的に定義（e.g.ディスクパック、シリンダetc）される。

b）セット型とセットオカーランス

(1) 論理構造

セット型Sは、レコード型$\underline{R}_1$と$\underline{R}_2$との間に存在して、これらの間の1対nの関係性を表わし、$\underline{S}(\underline{R}_1、\underline{R}_2)$と表わされる。$\underline{R}_1$と$\underline{R}_2$は、各々セット型Sの親及び子レコード型と呼ばれる。

セット型の外延、セットオカーランス集合Sは、レコードオカーランス集合$\underline{R}_1$と$\underline{R}_2$とに対する関係として定義される。

$$S \subseteq R_1 \times R_2 \quad （ここで、R_1 \neq R_2）$$

各Sに対して、以下の関数を定義する。

$$\forall r_1 \in R_1 \quad card(fs(r_1)) \geq 0$$

$$\forall r_2 \in R_2 \quad card(f_s^{-1}(r_2)) \leq 1$$

この時、R_1とR_2とは、各々Sの親レコードオカーランス集合、子レコード集合と呼ばれる。これを各々、owner(S)=R_1、member(S)=R_2と記す。

セットオカーランス集合Sの特徴は、次の様である。

(1) Sは、R_1とR_2との間の1対Nの関係性を表わす。

$$\forall r_1 \in R_1 \quad card(fs(r_1)) \geq 0$$

$$\forall r_2 \in R_2 \quad casd(fs^{-1}(r_1)) \leq 1$$

ここで、集合Aに対して、card(A)はAのカーディナリティである。又、owner(S)=R_1、memben(S)=R2とする。

(2) R_1のレコードオカーランスr_1($\in R_1$)に対するSの子レコードオカーランスの集合は順序づけられている。

$$\forall r_1 \in R_1 \quad fs(r_1) = \{ r_{2i} \in R_2 \mid i = 1, \cdots, m \}$$

この時、次の順序関係$\prec$に対して、$r_{2i_1} \prec r_{2i_2} \prec \cdots \prec r_{2m}$ である。tをR_2のあるデータ項目の集合({ tk_1、…tkh })とすると、

$$r_{2i} \prec r_{2j} \quad iff \quad r_{2i}[t] \leq r_{2j}[t] \wedge i \theta j$$

ここで $\prec = \begin{cases} \leq & \underline{if} \prec \text{は昇順である} \\ \geq & \underline{otherwise} \prec \text{は降順である} \end{cases}$

$r_{2i}[t] > r_{2j}[t]$の意味は、次の様になる。

$$r_{2i}[t] > r_{2j}[t] = \begin{cases} true & \underline{if} \quad r_{2i}[t_{k_1}] > r_{2j}[t_{k_1}] \\ & \vee (r_{2i}[t_{k_1}] = r_{2j}[t_{k_1}] \wedge r_{2i}[t_{2i}] = r_{2j}[t_{k_2}]) \\ & \vdots \\ & \vee (r_{2i}[t_{k_{m-1}}] = r_{2j}[t_{h_{m-1}}] \wedge r_{2i}[t_{hm}] = r_{2j}[t_{hm}]) \\ false & \underline{otherwise} \end{cases}$$

これは、ソートにおいてr_{k1}、…、t_{kn}が、左から右にmajor-miniorの役目を持つことを示している。この時、データ項目の順序集合tをソート項目と呼び、sort-item(S)=tと記す。又、sort-type(S)=ソートタイプ(∈{A(scending)、D(escending)})とする。

sort-item(S)が存在しない時、fs(r1)内の$r_{2i} \in R2$は、DBMSが自動的に順序づける。

$r_{1i} \in R_1$とfs(r_1)との順序集合は、特に、r_1を親レコードオカーランスとするセットオカーランスと呼ばれる。これを、次の様に表す。

$$s_0(r_1) = \{ r_1 \prec r_{2i} \prec \cdots\cdots \quad r_{2m} \}$$

(3) r_1のSに関する子レコードオカーランスに対して、これらのレコードオカーランス内

で、あるデータ項目の値が一意で（DNA：Duplicates are not allowed）であることが出来る。このデータ項目をｔとする。

$$\forall r_1 \in R_1\ \forall r_2, r_2' \in fs(r_1)\quad r_2[t] \neq r_2'[t]$$

$$\underline{iff}\quad r_2 \neq r_2'$$

この時、R2のデータ項目（集合）ｔを、セットオカーランス集合Ｓに関するDNA-ｉｔｅｍ（Ｓ）＝ｔと記す。

(4) Ｒ1が、ＳＹＳＴＥＭと呼ばれ、一つのレコードオカーランス（ｒ1）から成るレコードオカーランス集合の時、Ｓを単項セットオカーランス集合と呼ぶ。

(2) 物理構造

セットオカーランスは、１つの親レコードオカーランスと複数の子レコードオカーランスをポインタによって結び付けることにより実現されている。このポインタをたどってレコードオカーランスをアクセスする事ができる。

セットオカーランス内の子レコードオカーランスの格納順序は、セットオカーランスの先頭、末尾あるいはデータ項目の値によるものの３種がある。又、セットオカーランスの格納では、親レコードオカーランスの物理的近傍に子レコードオカーランスを格納するクラスタリングの方法もある。

ポインタをたどるパターン（アクセスパターン）には次の３つがある。

イ）順序集合の順序に基づいて、アクセスする。

ロ）順序集合の順序の逆でアクセスする。

ハ）子レコードオカーランス（r_{2i}）から、その親レコードオカーランス（ｒ1）をアクセスする。

この各々に対応して、セットオカーランス集合Ｓは、次の三種のポインタによって実現される。

イ）ＮＥＸＴポインタ

ロ）ＰＲＩＯＲポインタ

ハ）ＯＷＮＥＲポインタ

ここで、Ｓ内のレコードオカーランスｒに対して、次の記法を定義する。

ｎ－ｐｔｒ（Ｓ、ｒ）＝ＮＥＸＴポインタの指すレコードオカーランス

ｐ－ｐｔｒ（Ｓ、ｒ）＝ＰＲＩＯＲポインタの指すレコードオカーランス

ｏ－ｐｔｒ（Ｓ、ｒ）＝ＯＷＮＥＲポインタの指すレコードオカーランス

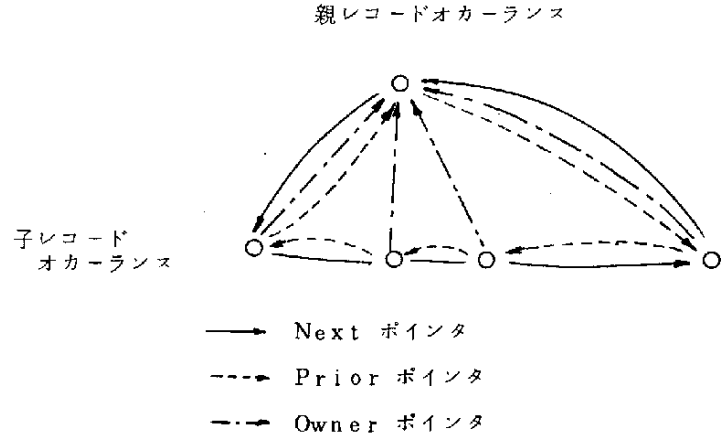


図3.1　next,prior,ownerポインタ

$$
\begin{aligned}
&\text{next} \quad n\text{-}ptr\ (S、ri) = r_{2\,1} \\
&\qquad \begin{cases} h\text{-}ptr\ (S、ri) = r_{i+1} & \text{if} \quad i \leq m-1 \\ n\text{-}ptr\ (S、ri) = r_1 & \text{if} \quad i = m \end{cases} \\
&\text{prior} \quad p\text{-}ptr\ (S、ri) = r_{2\,m} \\
&\qquad \begin{cases} p\text{-}ptr\ (S、ri) = r_{i-1} & \text{if} \quad i \geq 2 \\ p\text{-}ptr\ (S、ri) = r_1 & \text{if} \quad i = 1 \end{cases} \\
&\text{owner} \quad o\text{-}ptr\ (S、ri) = r_1 \qquad \text{for } i = 1、\cdots、m
\end{aligned}
$$

B．アクセス言語（DML）

CODASYLモデルのアクセス言語は、レコード単位のアクセスを行うものである。レコードオカーランスは、セットオカーランス内の順序をたどりながら巡航的（navigational）にアクセスされる。この為、現在どのレコードのレコードオカーランスがアクセスされているかが重要となる。

a）現在子

現在子（currency indicator）は、データベース内で一番最近にアクセスされたレコードオカーランスが何であるかを表わしている。現在子としては、次の4種がある。

i）実行単位（run-unit）の現在子（currency of run-unit）

ii）各エリアAの現在子　　（currency of A　　）

iii）各レコードオカーランス集合Rの現在子（currency of R　）

iv）各セットオカーランス集合Sの現在子　（currency of S　）

i）は、現在の実行単位（例えば、アプリケーションプログラムの実行）において、一番最近にアクセスされたレコードオカーランスを指している。これを次の様に表わす。

$$current = \begin{cases} r & \underline{if} \quad r\text{はデータベース内で一番最近にアクセスされたレコードオカーランスである。} \\ \perp & \underline{otherwise} \quad \text{まだ、どのレコードオカーランスもアクセスされていない。} \end{cases}$$

ii）は、各エリア（A）毎に保持され、エリア内のレコードオカーランスの中で一番最近にアクセスされた事を指している。これを次の様に表わす。

$$current(A) = \begin{cases} r \in A & \underline{if} \quad \text{エリアA内の一番最近にアクセスされたレコードオカーランスである。} \\ \perp & \underline{otherwise} \quad \text{まだ、エリア内のどのレコードオカーランスもアクセスされていない。} \end{cases}$$

iii）は、各レコードオカーランス集合R毎に保持され、R内のレコードオカーランスのなかで、一番最近にアクセスされたレコードオカーランスを指している。これを次の様に表わす。

$$current(R) = \begin{cases} r \in R & \underline{if} \quad r\text{はR内の一番最近にアクセスレコードオカーランスである。} \\ \perp & \underline{otherwise} \quad \text{まだ、レコード型Rのどのレコードオカーランスもアクセスされていない。} \end{cases}$$

iv）は、各セットオカーランス集合Sのセットオカーランスを指している。このレコードオカーランスを含んでいるセットオカーランスを、現在Sセットオカーランス（current S set-occurrence）と呼ぶ。これを次の様に表わす。

$$current(S) = \begin{cases} r \in R & \underline{if}\ r \in SO(r1)\text{は、現在Sセットオカーランスで、rはこの中で一番最近にアクセスされたレコードオカーランスである。} \\ \perp & \underline{otherwise} \quad \text{まだ、セットSのどのレコードオカーランスもアクセスされていない。} \end{cases}$$

b）順序関係のアクセス

CODASYLデータベースは、現在子の指すレコードオカーランスから、セットオカーランス内、エリア内、またはレコードオカーランス集合内の論理的な順序をたどることによりアクセスされる。ここでは、セットオカーランス内の論理的順序関係のアクセスを考える。ここで、Sをセットオカーランス集合、R1とR2とを各々Sの親と子レコードオカーランス集合とする。

この時、ある親レコードオカーランスr1のセットオカーランスSO（r1）において全てのr∈SO（r1）に対して、次の3つのアクセスを定義する。

$$next(S, r) = \begin{cases} r' & \underline{if} \quad r < r' \wedge \sim(\exists\, r''\ s.t.\ r < r'' < r' \\ \perp & \underline{otherwise} \end{cases}$$

$$\text{prior}(S, r) = \begin{cases} r' & \underline{\text{if}}\ \ r' < r \wedge \sim(\exists r''_{s.t.}\, r' < r'' < r) \\ \perp & \underline{\text{otherwise}} \end{cases}$$

$$\text{owner}(S, r) = \begin{cases} r' & \underline{\text{if}}\ \ r \in \text{fs}(r') \\ \perp & \underline{\text{otherwise}} \end{cases}$$

c) 直接アクセス

直接アクセスとしては、CALC(R、V)={ r | r∈R∧r〔t〕=v }がある。ここでRはレコードオカーランス集合、tはCALC項目(集合)である。この集合は、レコードオカーランスのdbキーの値の昇順に順序づけられている。

d) アクセス言語

CODASYLモデルのアクセス言語のもつアクセス方法としては、次の2種がある。

(1) 直接アクセス

(2) 順次アクセス

(1) 直接アクセス

直接アクセスは、次の3種類の文によってなされる。

(i) t←v； find any R.

この意味は、次の様である。

```
if  t=CALC-item(R)、then
    begin
        current←db-key(r)；s.t.  db-key(r)  is
                                the smallest in CALC(R、r)；
        current(A)←current(R)←current
    end；
```

(ii) find duplicate R；

tの値がvであるレコードオカーランスが複数存在する時、上記の文を用いてアクセスされる。

この意味は次の様である。

```
if  t=CALC-item(R) ∧ CALC-item(R)=DA、then
    begin
        〔current〔t〕=r〔t〕〕
        current(A)←current(R)←current←r；
                s.t. db-key(r∈CALC(R、r))が、
                    db-key(current)の次に大きいr
    end；
```

(iii) find R；DB-KEY is x.

この意味は次の様である。

current←current←(R)←current(A)←x;

変数xには、db-keyの値が後述するaccept文によってセットされる。

(2) 順次アクセス

順次アクセスは、論理的なセットオカーランス内の巡航を行う。

(i) find next R within S.

この意味は次の様である。

if R=member(S)∧current(S)≠⊥、then

begin

current←next(S、current(S));

current(A)←current(S)←current

end

next(S、r)は、常に物理的にサポートされる。

(ii) find prior R within S.

この意味は次の様である。

if R=member(S)∧current(S)≠⊥、then

begin

current←prior(S、current(S));

current(A)←current(S)←current

end

prior(S、r)が物理的にサポートされていないならば、一度親レコードオカーランスに戻り、イ)のnextポインタにより、rの一つ前まで達する必要がある。

(iii) find owner within S.

この意味は次の様である。

if current(S)≠⊥∧current(S)∈member(S)、then

begin

current←owner(S、current(S));

current(A)←current(owner(S))←current

end

owner(S、r)が物理的にサポートされていないならば、nextポインタ又はpriorポインタを用いて、親レコードオカーランスに戻らなければならない。

(iv) find first R〔within A realm〕.

この意味は次の様である。

current←r'∈R s.t. r'は、A内でdbキーの値が最小のレコードオカーランス

current(A)←current(R)←current;

(v) find next R〔within A realm〕.

この意味は次の様である。

current←r'∈R　s.t. r'は、A内でdbキーの値がcurrent(R)の次に大きいRのレコードオカーランス

current(A)←current(R)←current;

(v) find duplicate R within S using t.

この意味は次の様である。

```
if t=DA-item(S) then
  begin
    〔current〔t〕=r〔t〕
     current(R)←current←r
          s.t. r∈next(S、current(s))〕
    current(A)←current(S)←current
  end
```

(vi) find n R within A realm

ここでnは整数を表わす。この意味は次の様である。

current←r'∈R　s.t. r'はA内でdbキー値がn番目に小さいレコードオカーランス。

current(A)←current(R)←current;

現在子の値を取り出す為には、次のaccept命令を用いる。

(vii) accept x from currency

この意味は次の様である。

x←current;

(ix) accept x from R currency.

この意味は次の様である。

x←current(R);

(x) accept x from S currency.

この意味は次の様である。

x←current(S);

(xi) accept x from A currency

この意味は次の様である。

x←current(A)

(c) レコードオカーランスの獲得

レコードオカーランスは、次のget命令によって、ユーザ作業領域(UWA)に読み込まれる。

(1) get;

この意味は次の様である。

UWA←(current)；

ここで(current)は、currentの示すレコードオカーランスの内容である。

(2) get R；

この意味は次の様である。

UWA←(current(R))；

UWAに読み込まれたデータは、アクセス言語(DML)の親言語(例えば、COBOL、PL/I)によって、必要な処理がなされる。UWA内の処理結果に対して、CODASYLモデルのアクセス言語によって再びアクセスを行うことが出来ない。リレーショナルモデルの閉包性をCODASYLモデルは有していない。

3.2.2 スキーマ変換アルゴリズム

ここでは、CODASYLスキーマからリレーショナルスキーマへの変換について論じる。

A. 変換目標

スキーマ変換では、データベースが持っている情報内容を保存されねばならない。ここではCODASYLスキーマとリレーショナルスキーマとが互いに情報内容を保存しているとは、各々のモデル要素が互いに1対1に対応していることと定義する。

B. 変換アルゴリズム

CODASYLスキーマとリレーショナルスキーマとの対応を考える場合、主キー概念の対応づけが重要である。リレーショナルスキーマにおける主キーは、リレーションの組を一意に識別する属性又は属性集合である。又、各リレーションは必ず主キーを持っている。

これに対して、CODASYLスキーマでは、レコードオカーランス集合内の各レコードオカーランスを一意に識別できるデータ項目又はその集合は、一般に存在しない。この為、我々はレコードオカーランス集合内の個々のレコードオカーランスを一意に識別する機構として、dbキーを用いることにする。dbキーは、一つの実行単位内で、個々のレコードオカーランスを識別する値をとるが、ユーザにはこの値を視ることができない。

以下にCODASYLスキーマからリレーショナルスキーマへの変換アルゴリズムを示す。

a) レコード型R(t1、…、tn)に対して、リレーション$\underline{E(R)}$ ($\underline{@R}$、at_1、…、at_n)を生成する。

ここで、E(R)＝Rに対応したリレーション名

at_i ＝t_iに対応した属性名

@R ＝db-keyを取る属性で主属性と呼ぶ。@Rは主属性であるとともに、E(R)の主キーとなる。

b) セット型S(A、B)(但し、Bはリンクレコード型でない)に対して、リレーション$\underline{R(S)}$(@A、$\underline{@B}$)を生成する。

ここで、R(S)＝Sに対応したリレーション名

@A　＝リレーションE（A）の主属性

@B　＝リレーションE（B）の主属性

また、©Bはリレーション R（S）の主属性となる。

c）リンクレコード型 L（t_1、…、t_m）とセット型 Si（Ri、L）（i＝1、…、n）に対して、リレーション R（L）（p_1、…、p_n、at_1、…at_m）を生成する。

ここで、p_i＝E（R_i）の主属性

但し、p_iは、R_iと他のR_jが同一の場合、これを区別するためにS_iと@R_iの連結となる。その他の場合、p_iは@R_iとなる。

$at_i = t_j$に対応した属性名

a）で生成されたリレーションをEリレーションと呼ぶ。またb）とc）で生成されたリレーションは、関係性を表わしているので、Rリレーションと呼ぶ。Rリレーションの各主属性pに対して以下が定義される。

e－rel（p）＝pが関連づけるEリレーション名

b）の時、role（@A）＝Owner（親）、role（@B）＝Member（子）

c）の時、role（@Ri）＝Si。

リレーションRに対して、patt（R）をその主属性集合とする。又、レコード型、セット型、データ項目と、対応するリレーション、属性の名前とは、同一であるとする。

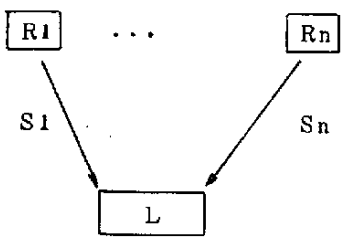


図3.2　リンクレコード型L

C．スキーマ変換例

例えば、図3.3は、プロジェクトとメンバ、メンバと論文、論文及びプロジェクトと、テーマ等の関係を表わした情報を有するCODASYLスキーマである。図3.5は、これから生成されるリレーショナルスキーマを示している。

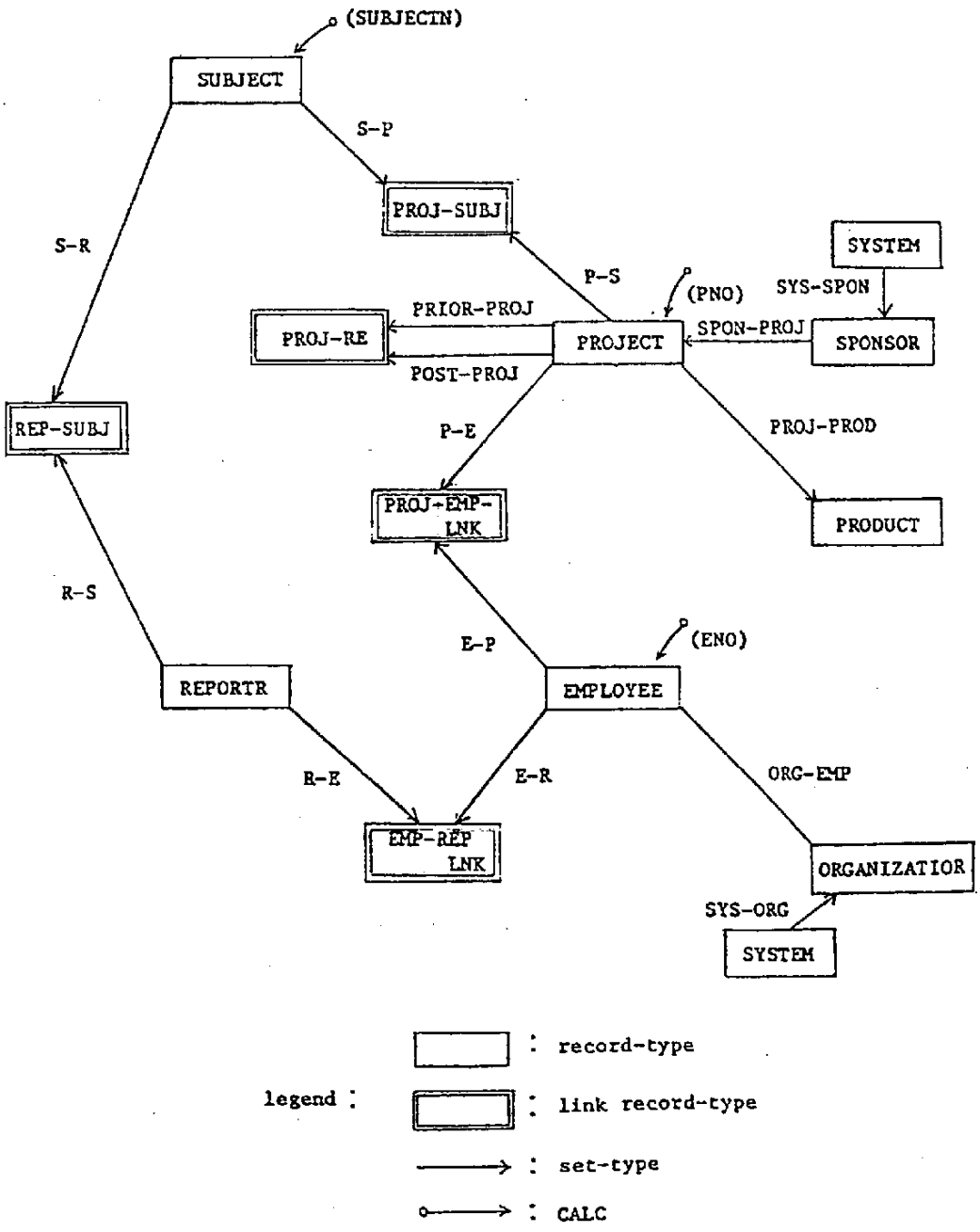


図3.3 PRDBSのスキーマ

```
 SCHEMA NAME IS      PROJJOBS  FOR DBT
      AUTHOR IS              M-SUZUKI
      DATE-WRITTEN           1980-08-16.
*     REMARKS                PROJECT DATABASE SYSTEM.
*                                          ****************
*                                          *  RECORD ENTRY  *
*                                          ****************
 RECORD NAME IS      EMPLOYEE.
      02  ENO                ;     PIC IS  9(05).
      02  ENAME              ;     PIC IS  X(10).
      02  EFNAME             ;     PIC IS  X(20).
      02  ESEX               ;     PIC IS  X.
      02  BIRTH-PLACE        ;     PIC IS  X(10).
      02  BIRTH-YEAR         ;     PIC IS  9(02).
      02  BIRTH-MONTH        ;     PIC IS  9(02).
      02  ALMA-MATER         ;     PIC IS  X(30).
      02  MAJOR              ;     PIC IS  X(30).
      02  DEGREE             ;     PIC IS  X.
      02  DEG-YEAR           ;     PIC IS  9(02).
      02  HIRED-YEAR         ;     PIC IS  9(02).
      02  RETIRED-YEAR       ;     PIC IS  9(02).
      02  DEPT               ;     PIC IS  X(20).
      02  EPOSITION          ;     PIC IS  X(20).
      02  TEL                ;     PIC IS  X(12).
      02  EXT                ;     PIC IS  9(04).
      02  FIL                ;     PIC IS  X(27).
*
 RECORD NAME IS      PROJECT;
      LOCATION MODE IS     RANDOM USING   PNO
           DUPLICATES ARE NOT ALLOWED.
      02  PNO                ;     PIC IS  X(08).
      02  PNAME              ;     PIC IS  X(100).
      02  PROJ-TYPE          ;     PIC IS  X.
      02  STATUS-P           ;     PIC IS  X(03).
      02  SYEAR              ;     PIC IS  9(02).
      02  EYEAR              ;     PIC IS  9(02).
      02  BUDGET             ;     PIC IS  9(11).
      02  FIL                ;     PIC IS  X(73).
*
 RECORD NAME IS      REPORTR;
      LOCATION MODE IS     RANDOM USING RNO
           DUPLICATES ARE NOT ALLOWED.
      02  RNO                ;     PIC IS  X(08).
      02  TITLE              ;     PIC IS  X(200).
      02  SOURCE-N           ;     PIC IS  X(100).
      02  VOL                ;     PIC IS  X(08).
      02  NUMB               ;     PIC IS  9(03).
      02  MONTH              ;     PIC IS  9(02).
      02  YEAR               ;     PIC IS  9(02).
      02  FROM-PAGE          ;     PIC IS  9(05).
      02  TO-PAGE            ;     PIC IS  9(05).
      02  TOTAL-PAGE         ;     PIC IS  9(05).
      02  NO-OF-AUTHOR       ;     PIC IS  9(02).
      02  LANGUAGE           ;     PIC IS  X(10).
      02  WRITTEN-YEAR       ;     PIC IS  9(02).
      02  COST               ;     PIC IS  9(09).
      02  RTYPE              ;     PIC IS  X.
      02  RSTATE             ;     PIC IS  X.
      02  FIL                ;     PIC IS  X(37).
*
 RECORD NAME IS      SPONSOR.
*          * NO CALC KEY *
      02  SNAME              ;     PIC IS  X(20).
      02  STYPE              ;     PIC IS  X(10).
      02  FIL                ;     PIC IS  X(70).
```

```
*
 RECORD NAME IS      SUBJECT;
      LOCATION MODE IS     RANDOM USING SUBJECTN
           DUPLICATES ARE NOT ALLOWED.
      02   SUBJECTN          ;     PIC IS  X(100).
      02   FIL               ;     PIC IS  X(100).
*
 RECORD NAME IS      PRODUCT.
*          * NO CALC KEY *
      02   PD-NO             ;     PIC IS  X(05).
      02   PD-NAME           ;     PIC IS  X(30).
      02   PTYPE             ;     PIC IS  X(01).
      02   COST              ;     PIC IS  9(09).
      02   YEAR              ;     PIC IS  9(02).
      02   LANGUAGE          ;     PIC IS  X(10).
      02   PSIZE             ;     PIC IS  9(05).
      02   OS-NAME           ;     PIC IS  X(15).
      02   FIL               ;     PIC IS  X(23).
*
 RECORD NAME IS      ORGANIZATION.
      02   ORG-NAME          ;     PIC IS  X(50).
      02   ORG-TYPE          ;     PIC IS  X(10).
      02   FIL               ;     PIC IS  X(40).
*
**********************
*  LINK RECORD TYPE  *
**********************
*
 RECORD NAME IS      PROJ-EMP-LNK.
      02   STATUSS           ;     PIC IS  X(04).
      02   SYEAR             ;     PIC IS  9(02).
      02   SMONTH            ;     PIC IS  9(02).
      02   EYEAR             ;     PIC IS  9(02).
      02   EMONTH            ;     PIC IS  9(02).
      02   PPOSITION         ;     PIC IS  X(10).
      02   RULE              ;     PIC IS  X(10).
      02   PERCENTAGE        ;     PIC IS  9(03).
*
 RECORD NAME IS      EMP-REP-LNK.
      02   AUTH-NO           ;     PIC IS  9(02).
*
 RECORD NAME IS      REP-SUBJ.
*
 RECORD NAME IS      PROJ-SUBJ.
*
 RECORD NAME IS      PROJ-RE.
```

図3.4　PADBSのレコード型定義

```
ESR SUBJECT (@J , subjectn)
    SPONSOR (@SPONSOR, sname,stype )
    PROJECT (@P, pno, pname, proj—type, status—f, syear, eyear,
                budget)
    PRODUCT (@PRODUCT, pd-name, ptype, cost, year, language, psize )
    EMPLOYEE(@E, eno efname, esex, birth-place, birth-year, birth-
                month, almater, major, degree, deg-year, hired,-year,
                retired-year, dept, position, tel, ext)
    REPORTR (@R, rno, title, source,-n, vol, numb, month, year, from-
                page, to-page, total-page, no-of-author, languag,
                written-year, cost, rtype, rstate )
    ORGANIZATIOR
                (@ORGANIZATIOR, org-name, org-type )
RSR SPON-PROJ(@SPONSOR, @P)
    PROJ-PROD(@P, @PRODUCT)
    PROJ-EMP-LNK
                (@P, @E, smonth, eyear, emonth, pposit, pposition,
                  role, percentage)
    PROJ-SUBJ(@P, @J )
    REP-SUBJ (@R, @J )
    EMP-REP-LNK
                (@R, @E, auth—no)
    PROJ-RE  (@P, prior—pr—@P)
    ORG-EMP  (@ORGANIZATIOR, @E)
```

図3.5 PRDBSのLCS

3.3 問合せ変換

3.2で変換されたリレーショナルスキーマ（LCS）に対してリレーショナル問合せ（これをLCS問合せと呼ぶ）によって検索を行う。LCS問合せは、LCSリレーションのみを参照するリレーショナル問合せである。このリレーショナル問合せを、CODASYLデータベースシステム上で実行可能なCOBOL DMLプログラムに変換し、実行させる必要がある。これを問合せ変換〔TAKIM80〕と呼ぶ。

問合せ変換の問題としては、次の2つがある。

・ データ構造（スキーマ）の相違

・ アクセス言語の相違

データ構造に関しては、CODASYLモデルはモデル要素であるレコード型がセット型によって関係づけられており、リレーショナルモデルはモデル要素であるリレーション間に関係性が存在しない。

アクセス言語に関しては、CODASYLモデルはレコード型間のリンクをたどりながらアクセスする手続的言語であり、リレーショナルモデルのリレーショナル計算言語（例えば、QUEL）は述語論理に基づいた非手続的言語である。

これらの問題点を検索に限定して解決し、LDP-V1.5として実現した。更新に関しては、第4章更新非手続的インタフェースLDP－V2で述べる。

ここでは、まず最初にLCS問合せを定義し、次にこのLCS問合せをCODASYL データベースシステムで実行可能なCOBOL DMLプログラムの生成について述べる。

3.3.1 ローカル概念スキーマ（LCS）問合せ

ローカル概念スキーマ（LCS）内のリレーションに対する問合せをLCS問合せと呼ぶ。LCS問合せは、内部的にはグラフで表現され、これをLCS問合せグラフと呼ぶ。

A. LCS問合せ

LCS問合せは、リレーショナル計算言語QUELを用いて、次の様に定義される。

range (l_1, L_1) … (l_n , L_n) ;

retrieve into R ($a_1 = ae_1$, …, $a_k = ae_k$)

where qual ;

L_i はLCSリレーション、l_i はその組変数である。l_i は、値として、L_i の組を取る。Rは結果リレーションで、a_j はその結果属性である。問合せの変数は、l.a（lは組変数、aはそのリレーションの属性）と記され、値として組lのa属性値の値を与える。ae_j は、条件式qualを満足する組 l_1、…, l_n 上の算術式で、結果属性 a_j の値を与える。各ae が参照する変数を目標属性と呼ぶ。

条件式qualは、制限及び結合述語から成る論理式である。組変数xに関する制限式rp(x)はQUELと同じである。組変数xとyの間の結合述語jp（x, y）には主結合述語と非主結合述語とがある。

a）制限述語 rp（x）::＝x.aθv | x.aθx.b

aとbは、xのリレーションの属性で、互いの定義域は等しい。θとしては、比較演算子{＜, ≦, ＝, ≠, ≧, ＞}の全てを取ることができる。

b） 主結合述語 pjp（x, y）::＝x.aθy.b

aとbは、各々xとyのリレーションの主属性で、互いの定義域は等しくなければならない。主結合は、通常の結合と異なり、事象間の意味的な関係性を表わしている。よってθとしては、＝と≠のみが許される。同様に主属性の算術計算は、行えない。 論理的には、

aggregate関数 count, ucount, anyのみが許される。

c）非主結合述語 njp(x, y)::=$ae_j(x)\ \theta\ ae_j(y)$

これは、通常の結合である。xとyに関する各々の算術式$ae_i(x)$と$ae_j(y)$の値が、$\theta \in \{<, \leq, =, \neq, \geq, >\}$について比較される。

QUEL〔HELDG75〕とLDP-V1.5で使用しているQUELとの相違点は表の様である。

表3.1 QUELとLDV−1.5のQUELの相違

項　目	QUEL	LDP-V1.5のQUEL
Range-Statement	RANGE of {Variable} IS Relation	RANGE ({Variable}, Relation);
Retrieve-Statement	RETRIEVE UNIQUE (target-list) 〔WHERE qual〕	対応するものが無い
Target-list	Variable. all	同　上
Comments	/* comment */	同　上
constant	String constants Integer constants Floating constants	String constants Integer constants
Functional expression	abs(n), ascii(n), ata(n) concat(a, b), cos(n), exp(n), gamma(n), log(n) mod(n, b), sin(n), sgrt(n)	abs(n), mod(n, b) （但し、問合せ変換では実現されていない。）
Aggregate expression	count, countu, sum, samu avg, avgu, max, min, any	同　左 （但し、問合せ交換では実現されていない。）
Pattern matching characters	パターンマッチング用の文字（*, ?, 〔〕）がある。	対応するものが無い。

d）条件式

条件式は、次の様に正規化されているとする。

$$qual ::= c1 \wedge \cdots \wedge cn$$

$$c_i ::= rc_i(x) \mid jc_i(x, y)$$

$$rc_i ::= rp_i(x) \vee \cdots\cdots \vee rp_{ili}(x) \quad (i = 1, \cdots, n)$$

$$jc_i ::= pjp(x, y) \mid njp_{i1}(x, y) \vee \cdots\cdots \vee njp_{ili}(x, y) \quad (i = 1, \cdots, n)$$

ここで、xとyは組変数である。各ciは、同一組変数を参照する述語の和でなければならない。

xの制限式$rc_i(x)$の積をrf(x)と記す。xとyの結合式$jc_i(x, y)$の積をjf(x, y)と記すと、次の形式を有している。

$$jf(x, y) ::= pjp(x, y)\ [\wedge njf(x, y)] \mid njf(x, y)$$

njf(x, y)は、pjp(x, y)ではないjci(x, y)の積である。即ち、jf(x, y)は一つの主結合述語もしくは1つ以上の非主結合述語から成る。

例として、図3.3のＬＣＳに対する問合せ「プロジェクトの開始以前に雇用されたメンバが、第一著者として書いた論文のテーマと、プロジェクトのテーマとが異なっている論文とそのプロジェクト番号を求めよ」を考える。これは、ＱＵＥＬで次の様に記される。

```
RANGE (E, EMPLOYEE) (P, PROJECT) (R, REPORTR)
RANGE (J, SUBJECT) ;
RANGE (PJ, PROJ-SUBJ) (RJ, REP-SUBJ) ;
RANGE (PE, PROJ-EMP-LNK) (ER, EMP-REP-LNK) ;
RETRIEVE INTO ANTI010 (P. PNO, R. RNO)
WHERE
P. @P=PE. @P   AND   PE. @E=E. @E   AND
E. @E=ER. @E   AND
ER. @R=R. @R   AND   R. @R=RJ. @R   AND
P. @P=PJ. @P   AND   PJ. @J=J. @J   AND
J. @J NE RJ. @J   AND
P. SYEAR GE E. HIRED-YEAR   AND
ER. AUTH-NO=1 ;
```

図3.6 ＬＣＳ問合せ

B．ＬＣＳ問合せグラフ

上記した様に正規された問合せは、ＬＣＳ問合せグラフ（ＬＱＧ）と呼ばれるグラフによって表わされる。グラフによって表わすことにより、問合せの記述と理解が容易になる。

ＬＱＧは、問合せ内の組変数を表わす節点集合ＬＮと、結合式を表わす辺集合ＬＥとからなる。各節点ｘは情報として（ｘ，ｒｆ（ｘ），ｔｌ（ｘ））を有している。ｘは、節点名で、対応する組変数名である。ｒｆ（ｘ）は、ｘの制限式であり、ｔｌ（ｘ）はｘの目標属性集合である。Ｅリレーション、Ｒリレーションを表わす節点を、各々Ｅ節点、Ｒ節点と呼ぶ。

節点ｘとｙ間の辺は、結合式を表わしている。結合式の種類とに応じて、ｉ）主、ⅱ）主不等、ⅲ）非主の３種の辺がある。ｉ）とⅱ）は、共に主結合述語ｘ．ａθｙ．ｂを表わし、ｉ）はθが" ＝ "（等号）で、ⅱ）はθが" ≠ "（不等号）の場合である。ⅲ）は非主結合式ｎｊｆ（ｘ，ｙ）を示している。

ＬＱＧは、各節点と辺の表わす条件式の積を条件式とし、各節点ｘのｔｌ（ｘ）の和を目標属性集合とするＬＣＳ問合せを表わしている。

例として、図3.6のＬＣＳ問合せに対するＬＱＧは図3.7の様になる。

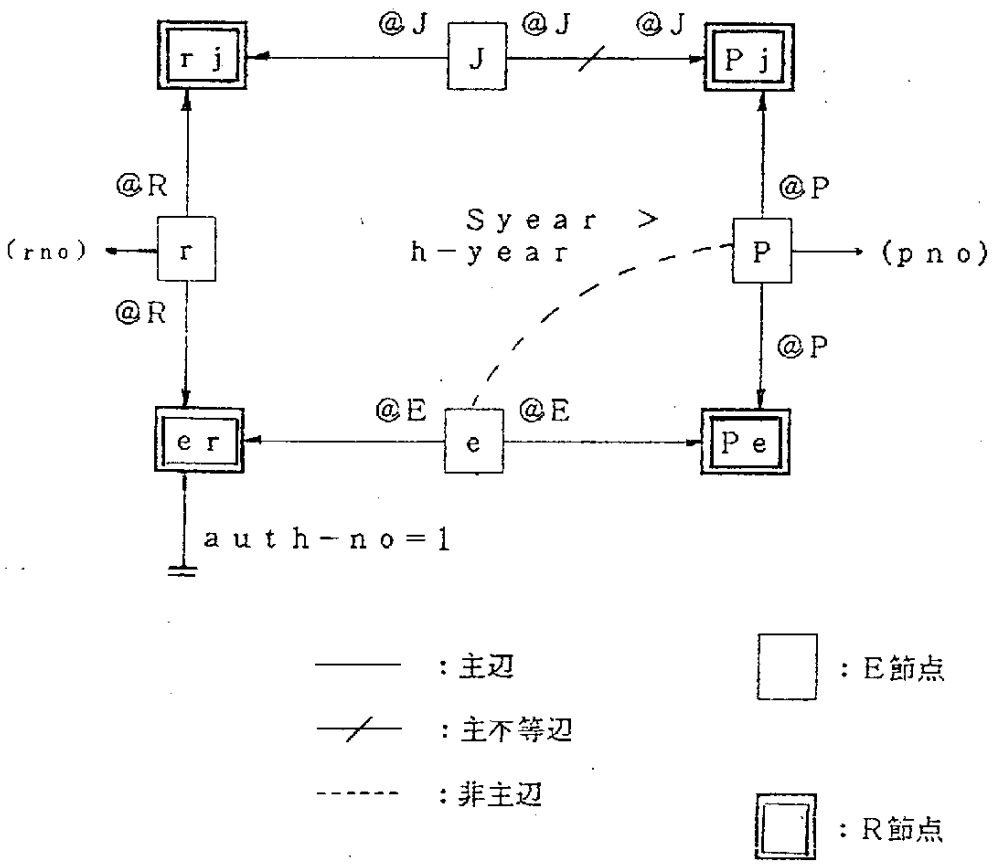


図3.7　ＬＱＧ

3.3.2 問合せ変換方法

問合せ変換の問題点としては、次の２つがある。

- データモデルの相違
- アクセス言語の相違

データモデルの相違は、ＣＯＤＡＳＹＬモデルではモデル要素レコード型間の関係性がセット型で表わされるのに対し、リレーショナルモデルではモデル要素リレーション間の関係性が存在しない事である。

アクセス言語の相違は、ＣＯＤＡＳＹＬモデルがセット型をたどりながらアクセスする手続的言語であるのに対し、リレーショナルモデルは述語論理に基づいた非手続的言語である。

第１の問題は、リレーショナルモデルにリレーション間の関係性を表わすリレーションを導し、ＱＵＥＬ問合せの意味をＣＯＤＡＳＹＬモデルによって非手続的に表わすことで解決する。リレーショナルモデル要素と要素間の関係性をＣＯＤＡＳＹＬモデルにの対応する要素と関係性に置き換える。この為の対応情報は、ＨＩ内に保持している。この情報を用いて、リレーショナル問合せ〔 3.3.1参照〕の意味をＣＯＤＡＳＹＬモデルによって表わすことができる(構造変換)。構造変換で生成されたＣＯＤＡＳＹＬモデルによる問合せは、ＣＯＤＡＳＹＬ問合せグラフ（ＣＱＧ）と呼ばれるグラフで表わされる。

第２の問題は、非手続的な問合せから手続を生成することで解決される。即ち、CODASYL問合せグラフ（ＣＱＧ）をＣＯＤＡＳＹＬデータベースシステムで実行可能な手続きに生成する。この手続生成をアクセスパス生成（ＡＰＧ）と呼ぶ。アクセスパス生成では、目標データベース上で最適なアクセス手続きを生成することが必要になる。この最適化の方法としては、

１）中間結果の量と数の最少化

２）応答時間の最少化

がある。この目標を達成する様に生成されるアクセス手続きは、アクセス木と呼ばれる木構造で表わされる。アクセス木から、自動的にＣＯＢＯＬ　ＤＭＬプログラムを生成する。これをＤＭＬ生成（ＤＭＬＧ）と呼ぶ。目標ＣＯＤＡＳＹＬ　ＤＢＳのＤＭＬ情報は、ＨＩ内のＤＭＬ情報（ＤＭＬＩ）に格納されている。このＤＭＬＩを、各ＣＯＤＡＳＹＬ　ＤＢＳ毎に生成することによって容易に各ＣＯＤＡＳＹＬ　ＤＢＳに適応できる。

生成されたＣＯＢＯＬ　ＤＭＬプログラムは、ローカル通信処理（ＬＣＰ）〔 3.3.9参照〕により、コンパイルから目標データベースの検索を行ない結果を出力する。

この問合せ変換の処理概要を示したのが図 3. 8である。ＬＱＧＰはＬＣＳ問合せからＬＱＧを生成するプロセッサ、ＣＱＧＰはＬＱＧをＣＱＧに構造変換させるプロセッサである。

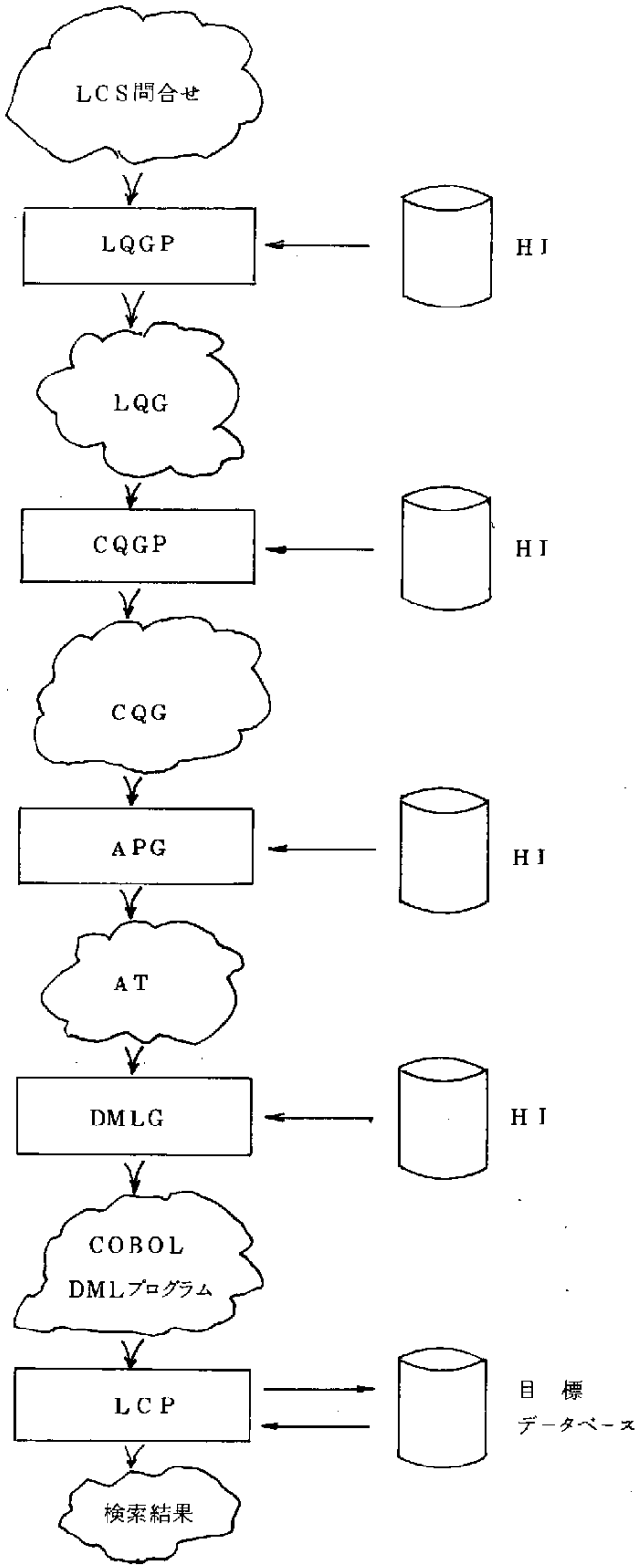


図3.8 問合せ変換の処理概要

3.3.3 問合せ構造変換

LCS問合せから、CODASYLモデルに基づいた非手続的な問合せを生成する。

A. CODASYL問合せ

CODASYLモデルに基づいた非手続な問合せを定義する。CODASYL問合せ(CQと記す)は、QUELと同様に次の様に記述される。

range (r_1 , R_1) … (r_m, R_m)

retrieve into R ($a_1 = ae_1$, …, $a_k = ae$)

where qual,

R_i はレコード型で、r_i はそのオカーランス変数である。r_i は、その値としてR_i 内のオカーランスを取る。Rは結果リレーション名で、a_j はその属性である。CQ内の変数は、r.a(rはオカーランス変数、aはそのレコード型のデータ項目)と記され、値としてオカーランスrのaの値を取る。ae_j は、条件式qualを満足するオカーランス r_1 , …, r_n 上の算術式で、結果属性 a_j の値を与える。各 ae_j 内の変数を目標項目と呼ぶ。

条件式qualは、制限と結合述語から成る論理式である。ここで、$\theta' \in \{<, \leq, =, \neq, \geq, >\}$, $\theta \in \{=, \neq\}$ とする。

a) 制限述語 $rp(x) ::= ae(x)\,\theta'\,v \mid ae_i(x)\,\theta'\,aej(x)$

ae(x)は、オカーランス変数xを参照する算術式である。例としては、図3.5のスキーマに対する問合せ「従業員番号が10以下の従業員名を求めよ」は、次の様に記述される。

range (e, EMPLOYEE) ;

retrieve into R (e, ename)

where e, eno ≤ 10 ;

b) 結合述語

これには、問合せの構造を表わす主結合述語と単に値の関連を表わす非結合述語がある。

(1) 主結合述語 $pjp(x, y) ::= sjp(x, y) \mid nsp(x, y)$

i) セット述語 $sjp(x, y) ::= S(x, y)$

Sはセット型で、xとyを各々Sの親、子レコード型とする。この時、sjp(x, y)は次の意味を持つ。

$$S(x, y) = \begin{cases} \text{true} & \text{if } (x, y) \in S \\ \text{fals} & \text{otherwise} \end{cases}$$

ii) 非セット述語 $njp(x, y) ::= ae_i(x)\,\theta'\,ae_j(y)$

これは通常の結合で、値の比較を行う。例としては、図3.5のスキーマに対する問せ「プロジェクト開始前に入社したメンバを求めよ」は、次の様に記述される。

range (e, EMPLOYEE)(p, PLOJECT);

retrieve into R (e. ename)

where P.s year ≥ e. hire-year ;

問合せの条件式は、ＬＣＳ問合せと同様に積正規化されている。但し、ＣＱでは、jc_i（x, y）は次の様である。

$$jc_i(x, y) ::= sjp(x, y) \mid nsp(x, y) \mid njp_{i1}(x, y) \vee \cdots \vee njp_{ii}(x, y)$$

ＣＱでは、結合述語として主結合又は非主結合述語から成り、主結合述語はセット述語か非セット述語のどちらかを取る。

セット述語の否定は、次の様に正規化される。

$$\sim S(x, y) ::= x \neq x' \wedge S(x', y) \mid S(x, y') \wedge y \neq y' \mid x \neq x' \wedge y \neq y' \wedge S(x', y')$$

セット述語の否定は、次の３つの可能性を持つことを意味する。1）セット型Ｓの親x'とｘが等しくない。2）セット型Ｓの子ｙ'とｙが等しくない。3）セット型Ｓの親x'とｘが、子ｙ'とｙが等しくない。

rf（x）もＬＣＳ問合せと同じである。jf（x, y）は、次の形式を持つ。

$$jf(x, y) ::= sjp(x, y) \wedge njf(x, y) \mid njf(x, y) \mid njf(x, y) \wedge nsp(x, y)$$

２節点（ｘとｙ）間の正規化された結合式としては、次の３つの形式になる。1）セット述語と非主結合述語の積。2）非主結合述語の積。3）非セット述語と非主結合述語の積。

例としては、図3.6のＬＣＳ問合せに対するＣＯＤＡＳＹＬ問合せは、次の様に記述される。

```
range  (e, EMPLOYEE)(p, PROJECT)(r, REPORTR);
range  (j, SUBJECT)(pj, PROT-SUBJ)(rj, REP-SUBJ);
range  (pe, PROJ-EMP-LNK)(er, EMP-REP-LNK);
retrieve into ANTIOJO (p.pno, r.rno)
where
P-E(p, pe) and  E-P(e, pe)  and
E-R(e, er) and  R-E(r, er)  and
R-S(r, rj) and  P-S(p, pj)  and
S-P(j, pj) and  ~S-R(j, rj) and
p.syear≧e.hire-year  and
e.auth-no=1;
```

B．ＣＱＧ

正規化された問合せは、ＣＯＤＡＳＹＬ問合せグラフ（ＣＱＧ）と呼ばれるグラフによって表わされる。ＣＱＧは、オカーランス変数を表わす節点集合ＣＮと、結合式を表わす辺集合ＣＥとから成る。ＣＮ内の節点ｘは、情報（ｘ，rf(x), tl(x)）を有している。ｘは節点名で対

応するオカーランス変数名である。rf(x)とtl(x)とは、各々xの制限式と目標属性集合である。

節点xとy間の辺は、結合述語の種類に応じて、i) セット、ii) 等、iii) 不等、iv) 非主の4種の辺から成る。各々、S(x, y)、x=y、x≠y、njf(x, y)を表わす。

CQGは、節点と辺が表わす条件式とし、各節点xのtl(x)の和を目標属性集合とするCODASYL問合せ(CQ)を表わしている。

例として、図3.8のLQGに対応するCQGは次の様になる。

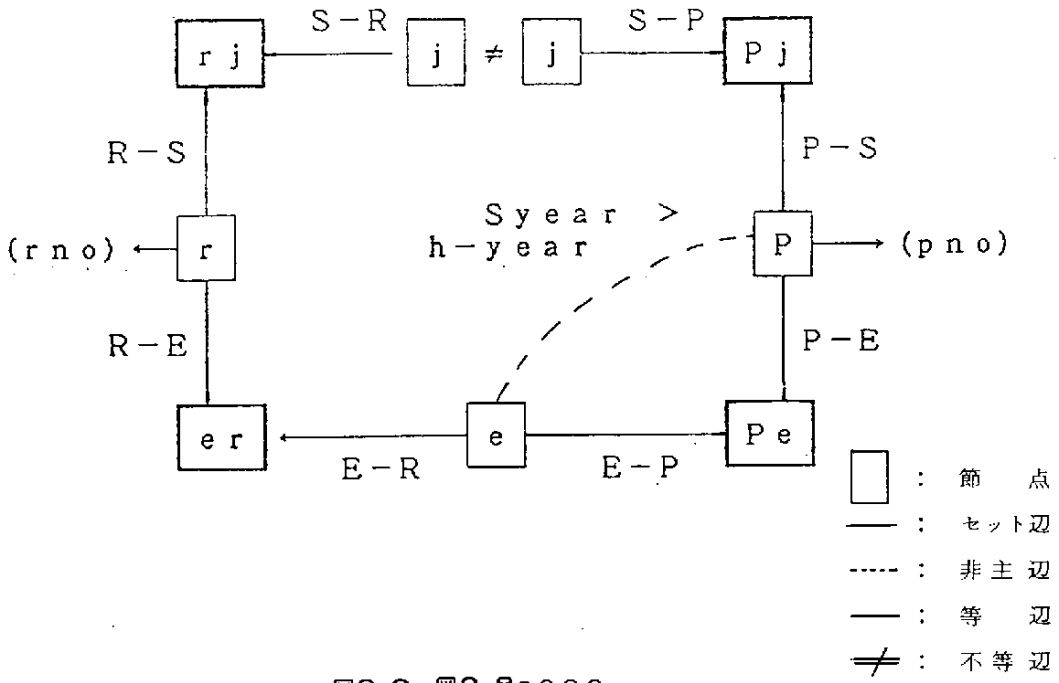


図3.9 図3.8のCQG

C. 構造変換

構造変換とは、LCS問合せからCODASYLモデルに基づいた非手続的問合せを生成することである。構造変換は、スキーマ変換の逆過程である。スキーマ変換で生成された異種情報(HI)を用いて、LQG内のリレーション要素(リレーションetc)をCODASYL モデル要素(レコード型etc)に置き換える必要がある。

E(R)リレーションはスキーマ変換において、CODASYL モデルのレコード型と1対1に対応づけられている。又、R(L)リレーションは、リンクレコード型に1対1に対応づけられている。問合せが参照するCODASYL 構造(アクセスパス)は、LCS問合せ内の主結合によって表わされる。従って、主結合は対応するセット型によって表わされる。

まず最初に、LCS問合せの構造を調べ、隠れ構造の検出と同一ノードの併合を行なう。この結果、LCS問合せの要素をCODASYL 問合せの要素に1対1に対応づけて変換が行な

える。

a）隠れ構造

ＬＣＳ問合せ内の主結合は、ＬＣＳリレーションの主属性に対してなされる。ある主属性のａは、Ｅ（Ｒ）リレーションと、これと他のＥ（Ｒ）リレーションとを関連づけるＲ(Ｓ)リレーション内に現われる。従って、主属性についての主結合としては、次の２つが考えられる。

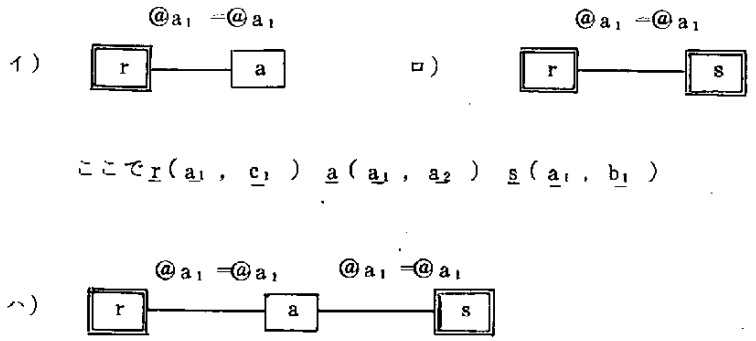


図3,10　ＬＱＧにおける主属性の主結合

図3.10のロ）は、意味的には、ハ）の様に解釈される。即ち、ある事象（オブジェクト）ａについての関係性ｒとｓとを示している。ユーザの発するＬＣＳ問合せは、図3.10のロ）の様に、Ｅ（Ｒ）リレーションを省略した形式で書ける。我々は、この様なＥ（Ｒ）リレーションを隠れ構造（ＨＳ）と呼ぶ。即ち、隠れ構造とは、ＬＱＧ内の主結辺がＲ（Ｓ）リレーションに対応する節点間に存在している時、Ｒ（Ｓ）リレーションの主結合属性を持つＥ（Ｒ）リレーションである。

ここで、ある主結合辺 pjl_{ij} を考える。主結合属性をａとする。ＬＱＧ上で、主結合辺で辺の推移関係を通して互いに連結な主結合辺の集合をＰＪＬａとする。

$$PJLa \triangleq \{\, pjl_{ij} \mid pjl_{ij} = r_i.a\ \theta\ r_j.a \wedge a \in pkey(r_j) \wedge a \in pkey(r_j) \,\}$$

ここで、ＰＪＬａ内の全ての主結合辺（ $pjl_{ij} \in PJLa$ ）は、連結である。

この時の隠れ構造を見つけるアルゴリズム（ＨＬＦアルゴリズム）を以下に示す。

① ａを主属性とするＬＣＳリレーションをＡとする。これは異種性情報（ＨＩ）内のＡＴＴリレーション（3.4.2のＢ.ＨＩを参照）を用いて得られる。

② ＰＪＬａ内の全節点 n_k をサーチして、値としてＬＣＳリレーションＡを取る n_k を捜す。

③ もし見つかれば、（ $n_k.a\ \theta\ r_l.a$ ）をＰＪＬａから消去（マーク）する。

もし見つからなければ、④へ。

次に、$n_k \cdot a\ \theta\ r_1 \cdot a$なる辺を見つけ、ＰＪＬａから消去する。

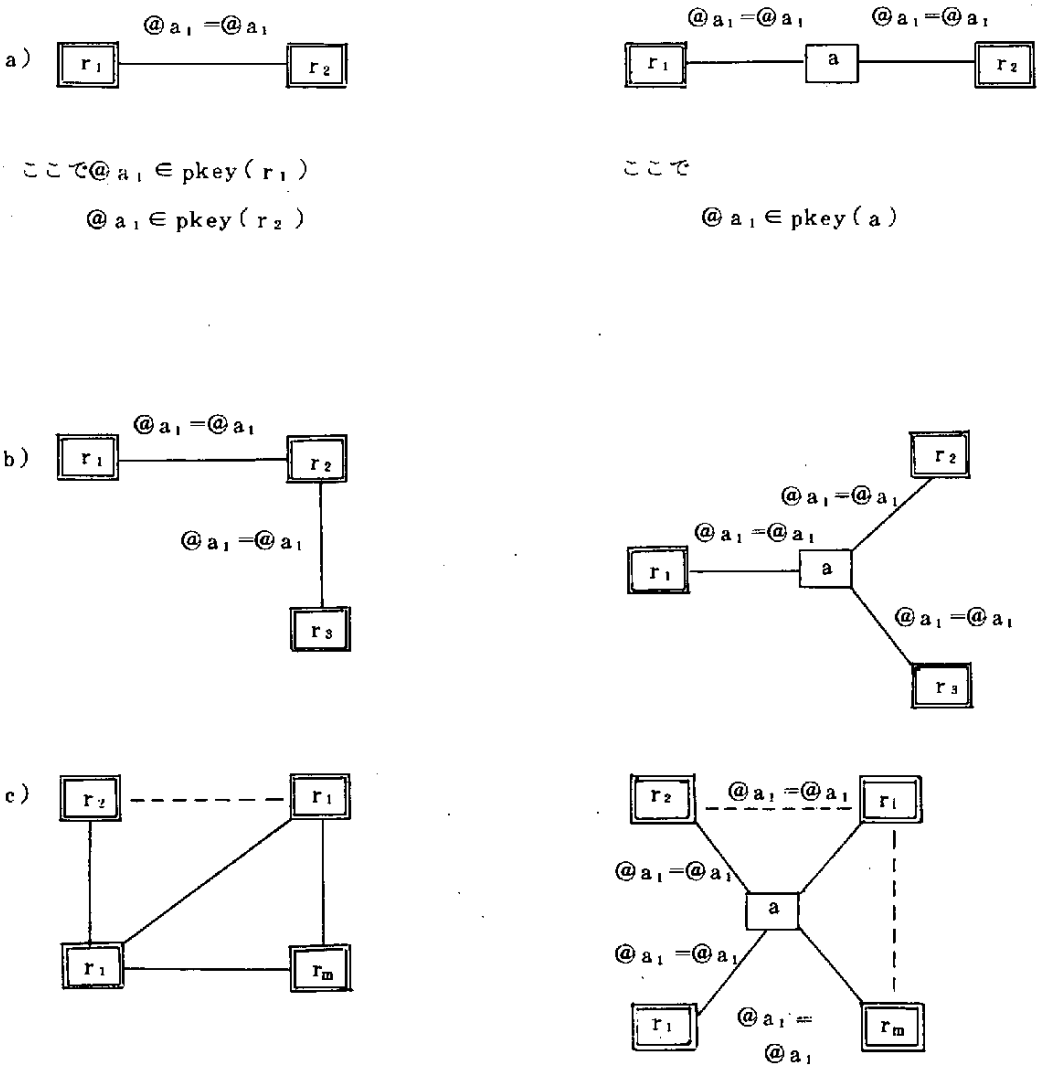


図3.11　等価主結合の隠れ構造の例

④ ＰＪＬａの辺を縦型にサーチする。マークされていない各辺$r_i.a\ \theta\ r_j.a$に対して、$r_i.a\ \theta\ A.a$と$A.a\ \theta\ r_j.a$の結合辺を生成し、ＰＪＬａから$r_i.a\ \theta\ r_j.a$を消去する。

⑤ ＰＪＬａ$=\varnothing$となるまで④を繰り返す。

ＰＪＬａ$\neq\varnothing$ならば、次の主属性について、①から繰り返す。

このアルゴリズムは、等価主結合（$\theta \in \{=\}$）に対して成り立つ。しかし、$\theta \in \{\neq\}$に対しては、隠れ構造の発見は、図3.12に示す様に、問合せの意味のあいまいさをもたするために出来ない。即ち、隠れ構造に対して、考えられる隠れ構造が複数存在し、一つに決定出来ない。この為、我々は、不等価主結合に対して、隠れ構造を許さないことにする。

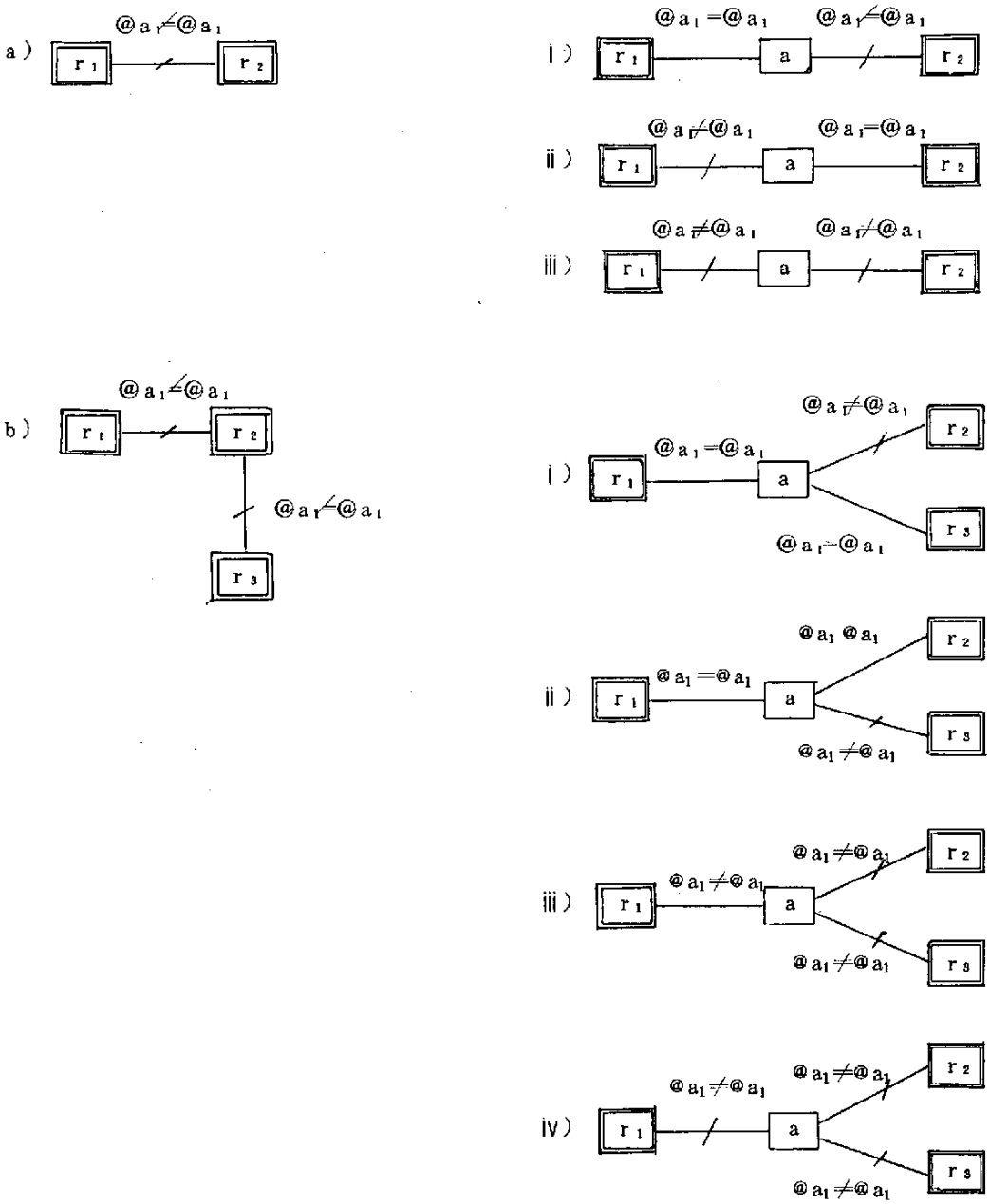
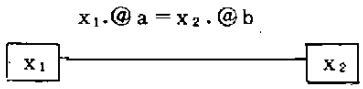


図3.12　等価主結合隠れ構造のあいまいさ

b）同一節点の併合

ＬＣＳ問合せ内の節点には、組変数名は異なるが意味的に同一組を表わしている節点が存在する。この為、ＬＣＳ問合せ内をサーチして、同一節点で節点間の併合を行なう。

まず最初に、同一節点の定義を行なう。同一節点とは、次の様なものをさす。



組変数x_1，x_2の参照するＬＣＳリレーションが同一で、そのＬＣＳリレーションがE(R)リレーションで、a、bがその主属性である時、x_1とx_2は同一節点である。

x_1とx_2 の併合は、次の図の様にする。

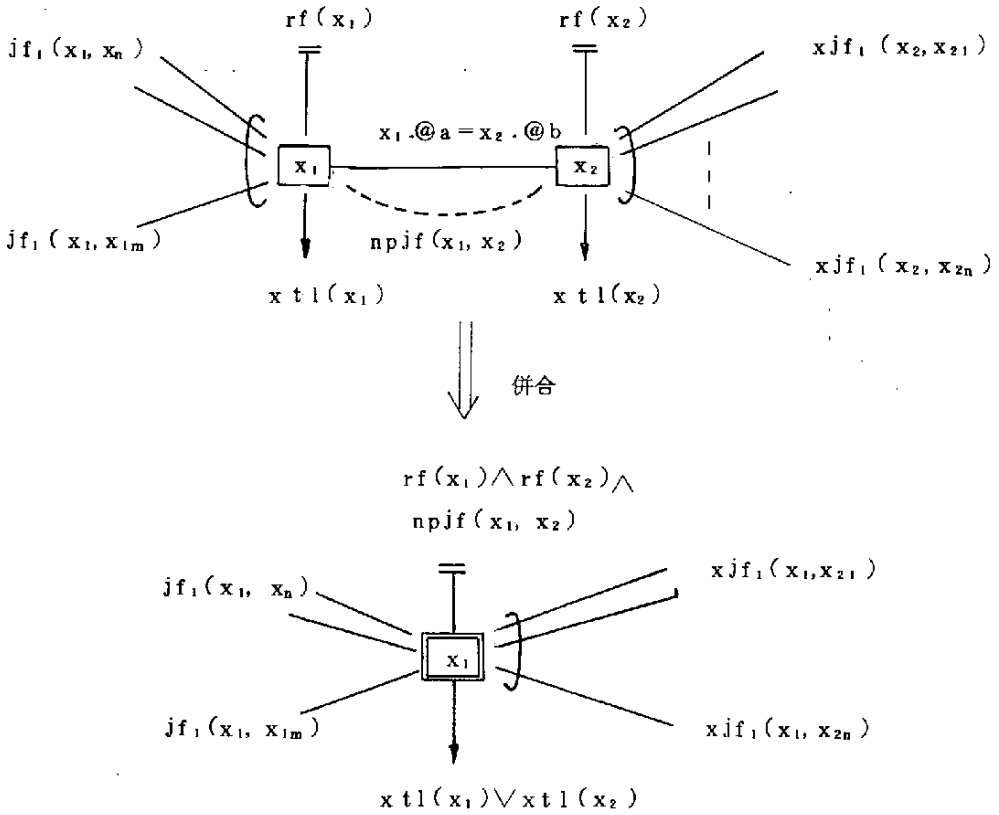


図３．１３　同一節点の併合

x_2の結合式内の組変数を全てx_1に置き換える。x_2の制限式rf(x_2)，x_1とx_2の非主結合式npjf(x_1，x_2)とx_1の制限式rf(x_1)の積をx_1の制限式とする。又、x_1，x_2の目標属性集合の和をx_1の目標属性とする。

c）構造変換

ＬＣＳ問合せからCODASYL問合せ（ＣＱ）への構造変換について論じる。

xとzをＬＱＧのE節点（Ｅ（Ｒ）リレーションを参照する組変数）、yをR節点（R(S)リレーションを参照する組変数）とする。また、aとdを各々xとzの主属性、bとcを各々xとzに対応するyの主属性とする。この時、構造変換アルゴリズムは、以下の様に表わせる。

① ＬＱＧ内の全てのE節点xからＣＱＧ節点xを作る。

② R節点yと関連する辺から、yに対応するCODASYLモデル要素yがセット型かリンクレコード型により、充の様にＣＱＧ辺を生成する。

ｉ） y＝セット型

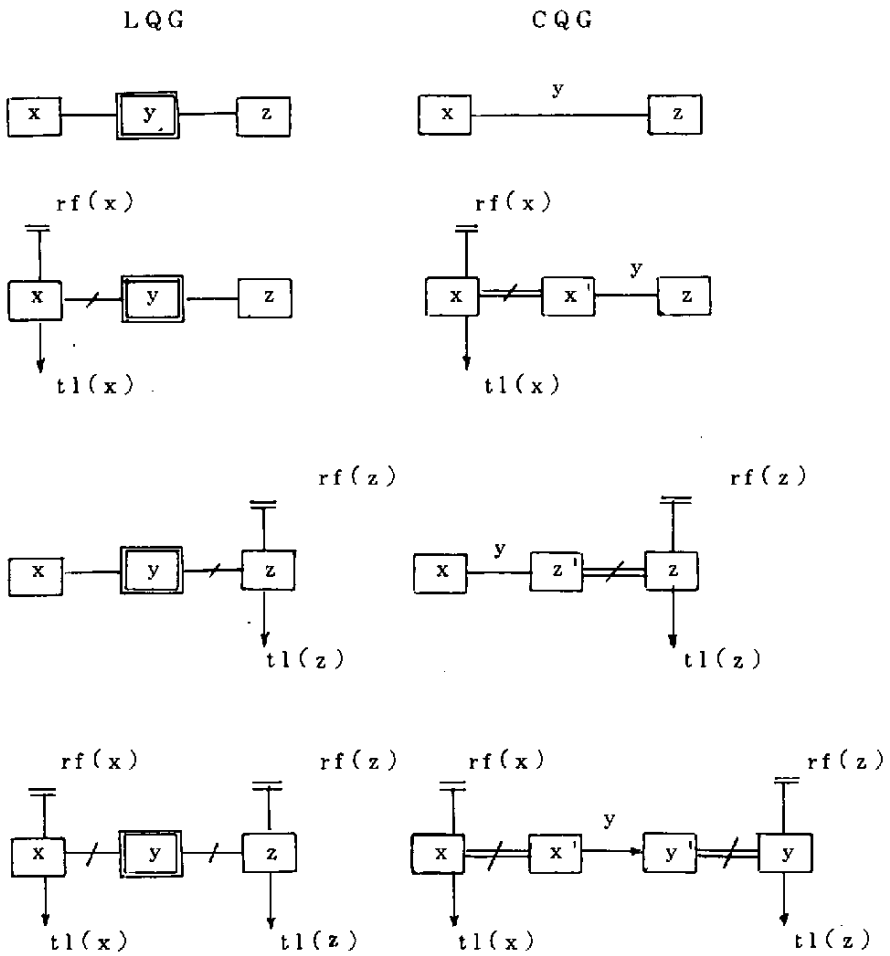


図3.14 構造変換（y＝セット型）

ⅱ） y＝リンクレコード型

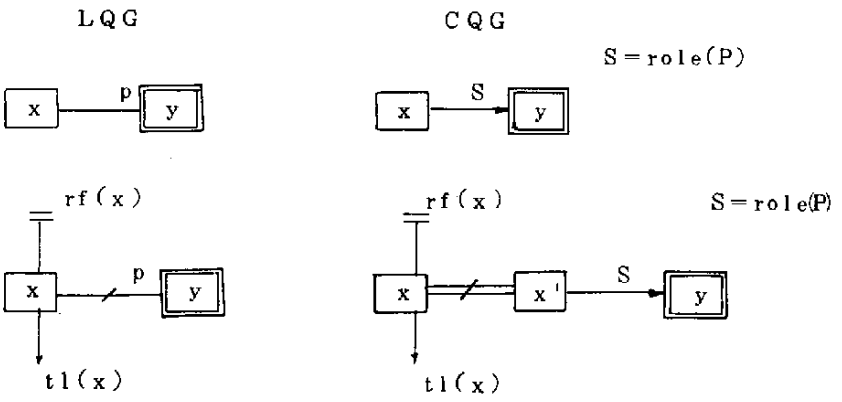


図3.15　構造変換（y＝リンクレコード型）

③　目標属性、制限式、非主結合式内の属性を対応するデータ項目に置換する。

構造変換では、E（R）リレーションはレコード型に、R（L）リレーションはリンレコード型に、1対1に対応づけられる。また問合せが参照するCODASYLモデルのデータ構造（アクセスパス）は、LCS問合せ内の主結合によって表わされる。従って、主結合は、1対1に対応するセット型によって表わされる。即ち、このアルゴリズムで生成されるLQGとCQGの表わす問合せの意味は等価である。

構造変換の例としては、図3.16のLQGは構造変換によって図3.16のCQGに変換される。

図3.16のE節点は全て対応するレコード型に変換され、LQGの主辺はCQGではセット辺に変換される。図3.16のR節点は全て対応するリンクレコード型に変換されている。LQGのJとPJの主不等辺は、CQGでは、JとPJのセット辺（S－P）及びJ間の不等辺に変換される。

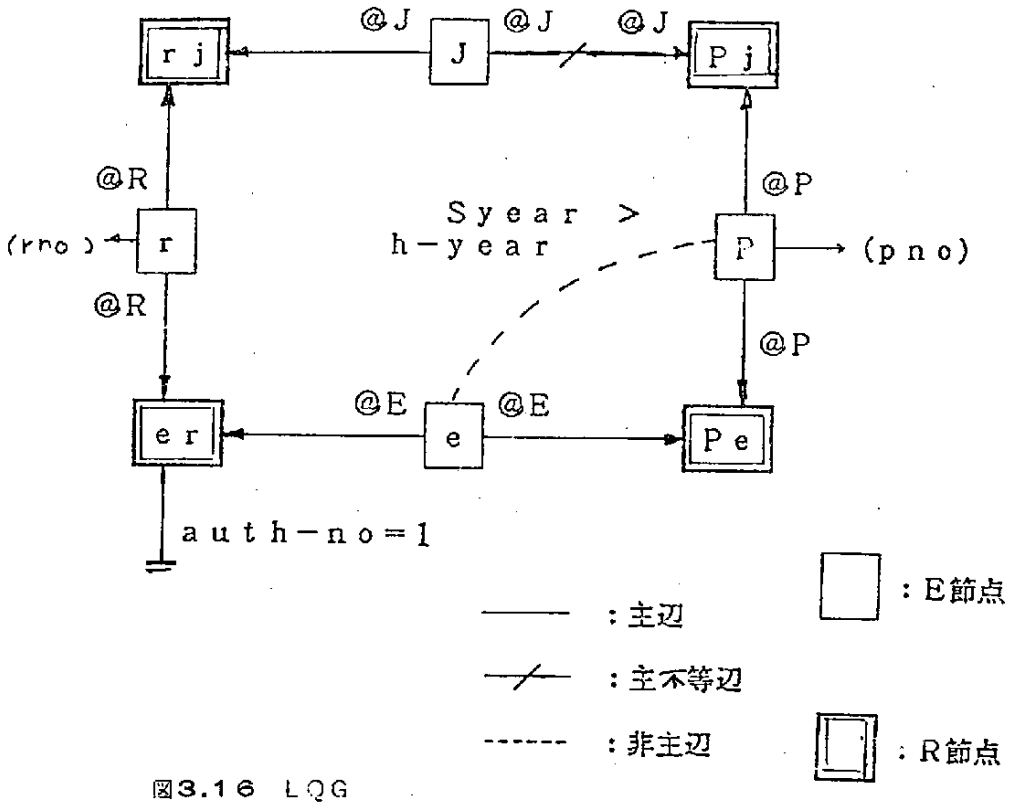


図3.16 LQG

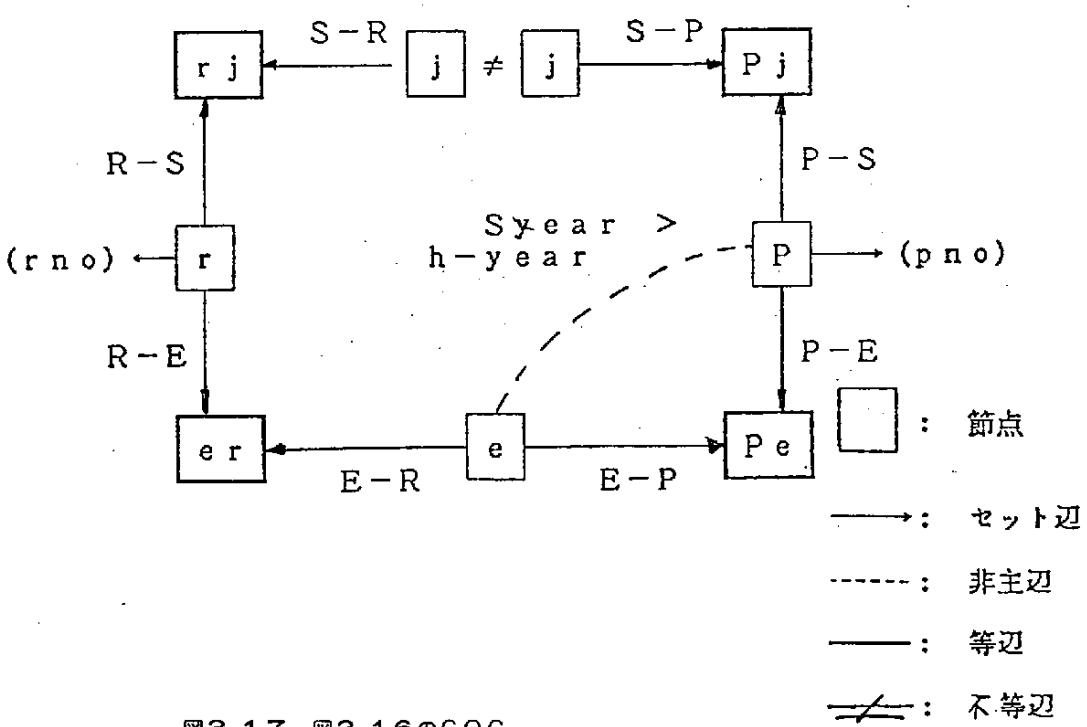


図3.17 図3.16のCQG

3.3.4 CODASYL問合せグラフ（CQG）の分割

等価主結合辺に対しては、CODASYLモデルに対応するアクセスパス（セット辺）が存在する。しかし、不等価主結合、非主結合辺に対しては、CODASYLモデルに対応するアクセスパスが存在しない。不等価主結合辺に対して非セット辺を対応させる。一般に、非セット辺や非主結合辺を処理する為には、不等価主結合式や非主結合式に対応した集合演算が必要となる。この為に、COBOL DMLプログラムで検索した結果を中間結果として、親言語で処理することが必要となる。

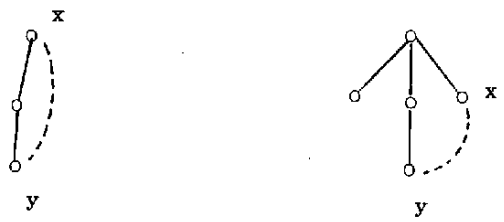
セット辺によって連結な節点に対しては、集合演算は不要である〔TAKIM80a〕ことから、CQGをサーチして連結な節点集合（GQGi）を見つけることが必要である。

連結な節点集合内の非主結合辺、非セット辺は、次の2種に分けられる。

1） supported

2） non-supported

1）は、非主結合辺（あるいは非セット辺）で結ばれた2つの節点（x，yとする）に対して、一方（xとする）が他方（y）の先祖である場合である。即ち、図のaの場合である。この場合合、非主結合辺に対しては、先にアクセスされたオカーランス（x*）の結合属性の値と他方（y*）



1） supported　　2） hon-supported

図3.17 非セット辺、非主結合辺のsupported／non-supported

の結合属性の値が結合式を満たすか調べれば良い。また、非セット辺の場合は先にアクセスされたオカーランス（x*）と、他方でアクセスされたオカーランス（y*）が異なっていることを確めれば良い。

2）に対しては、CQGiの結果リレーションを生成する時に結合式のチェックを行なう。この為、xとyの結合式か結合属性を各々の目標属性として加える。

連結な節点集合を見つけることは、次に示す様なCQG分割アルゴリズム（CQGD）によって行なえる。

CQG分割アルゴリズム

i） CQG内の非セット辺を一時的に切断する。

ii） CQG内のセット辺を、縦型にサーチして連結なCQG節点の集合のグループ（CQGi）をつくる。

iii) 分割された各ＣＱＧ節点グループ（ＣＱＧｉ）に対して結合スケジュールを作る。

iv) 各ＣＱＧ節点グループ（ＣＱＧｉ）に対して、〔TAKIM80a〕〔3.3.5〕によってアクセス木を生成する〔ＤＦＡ〕。

v) 各アクセス木からCOBOL DMLプログラムを生成する〔DMLG〕。

vi) 生成されたCOBOL DMLプログラムをCODASYL DBS上で実行し、結果を編集してユーザに出力する。

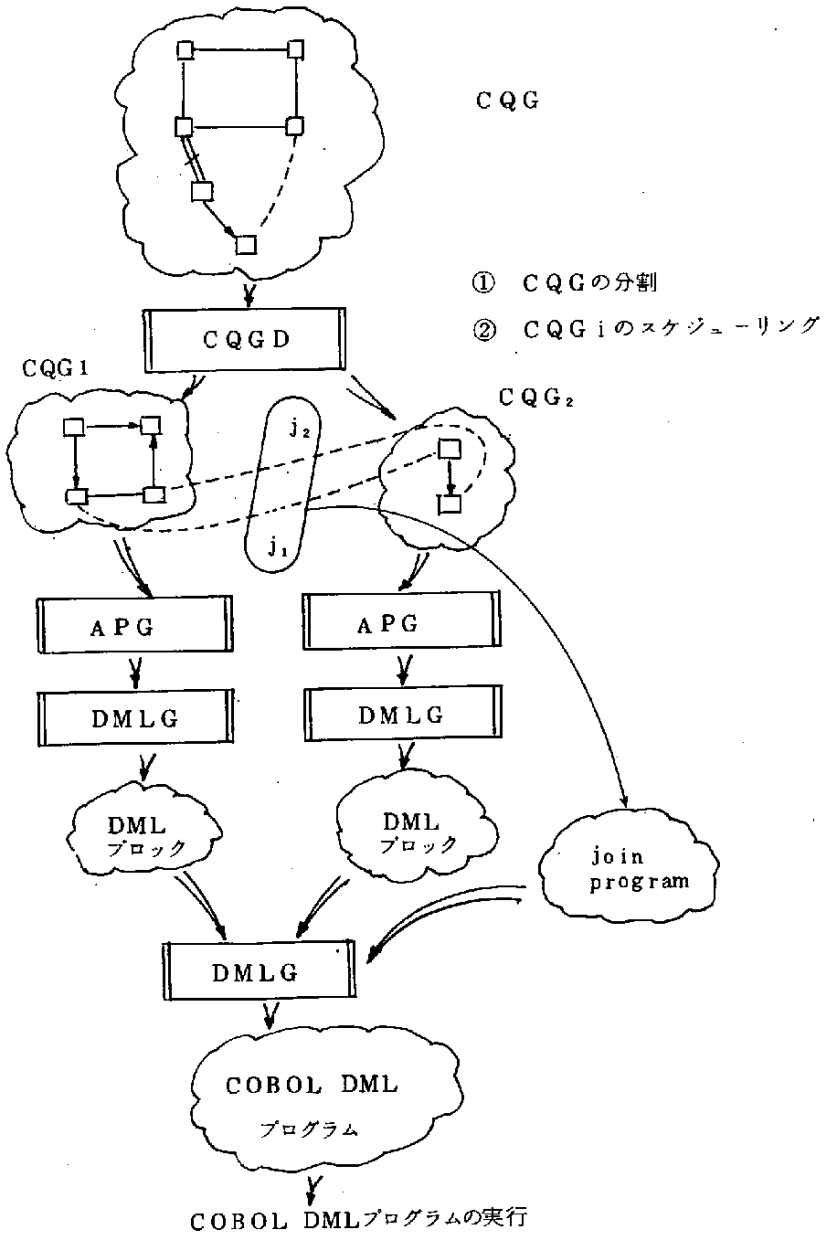


図3.18 CQGDの概要

3.3.5 アクセスパス生成（ＡＰＧ）

非手続的なＣＯＤＡＳＹＬ問合せから、オカーランス単位のアクセスを行う為のアクセスパスを生成することが必要である。

Ａ．目　標

アクセスパスは、次の目標を達成する様に生成される。

ｉ）中間結果の数と量の最少化

ii）アクセスされるオカーランス数の最少化

ＣＯＤＡＳＹＬ モデルでは、リレーショナルモデルと異なり、導出された結果を再度ＤＭＬによってアクセス出来ない。ＣＯＤＡＳＹＬ モデルが提供する機構を用いてアクセスを行う為には、一連のレコードオカーランス巡航によって結果を得る必要がある。巡航が断たれるとそこに中間結果が生成される。この数の増加は、格納と操作の為のファイル管理負荷の増大をもたらす。以上より、なるべく巡航を断たずに、ＣＯＤＡＳＹＬ モデルとは独立な中間結果の生成を減少する必要がある。

次に、生成されるＣＯＢＯＬ ＤＭＬプログラムの応答時間はアクセスされたオカーランス数に比例すると仮定する。実際の応答時間は、アクセスされるページ数に依存してくるが、このページアクセスはデータベースシステムの内部スキーマレベルの問題（ページに格納されるオカーランス数の問題）であると考えている。この仮定の正しさについては、評価において検討する。

Ｂ．ＤＦアルゴリズム（ＤＦＡ）

ＣＯＤＡＳＹＬ モデルにおけるアクセスパスは、セット型とレコード型を巡航しながらたどっていくものである。ＣＯＤＡＳＹＬ 問合せグラフ（ＣＱＧ）は、レコード型を節点とし、セット型を節点間の辺とするグラフ構造をしている。このグラフ表現から、アクセスパスを生成する為には、グラフ内の全てのセット型をユニークに含む木（これをアクセス木と呼ぶ）を生成しなければならない。ＣＱＧからアクセス木（ＡＴ）を生成するアルゴリズムは、ＣＱＧを縦型にサーチして生成するＤＦＡと呼ばれるものである。

ＤＦＡでは、よりオカーランス数の少ないものを、より早くアクセスするという簡単なヒューリスティックスを用いている。ここで、ｘとｙをＣＱＧ節点、ｌをｘとｙ間の辺とする。また、acc_n（ｌ，ｙ）は、ｘの次の節点ｙをｌ経由でアクセスした時の評価関数である。acc_nについては、Ｅ.評価関数で述べる。最初全てのＣＱＧ節点と辺は、Ｆ（ree）と印されているとする。ＣＱＧ節点ｘに、対応するＡＴ節点をｔ、その１つ上のＡＴ節点をｔｏとする。

ＤＦＡ

① Ｆ印の全てのＣＱＧ節点ｘを調べて、acc_n（－，ｘ）が最少のｘを見つける。
ｘが見つかった時、ｔｏ←Λ；ｐｕｓｈ ｄｏｗｎ（Λ，Λ）；
見つからなかった時は終了。

② ｘに対応したＡＴ節点ｔを作る。ｔｏ≠Λならば、ｔｏの一番右の子としてｔを結合する。

③　xがF印ならば、U（sed）と印し、pt(x)←t；

④　xがU印ならば、x，t，t'，(＝pt(x))をC（onflueut）と印す。

⑤　xがC印ならば、tをC印す。

⑥　CQG内のF印のセット辺の中で、accn（l，y）が最少のlとyを見つける。見つかったならば、lをCと印す。

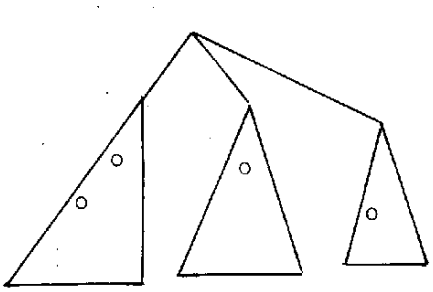
```
pushdown( x, t )；
to← t；
x ←y；
go to ②；
```

⑦　見つからない時、popup（x，t）

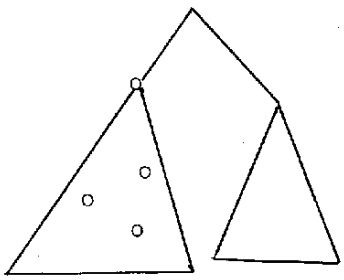
x≠Aならば、②へ

x＝Aならば、生成されたATに対してDMLを生成する。

生成されたアクセス木のAT節点はCQG節点と、AT技はCQG辺と各々1対1に対応する。しかし、DFAは、CQG内の辺をユニークにたどることから、CQG内にループが存在する時には、あるCQGセット型lに対して、複数のAT節点t_1，…，t_n（これらをc(x)と記す）が存在し得る。n＞2の時、c(x)内のAT節点をxの合流（cofluant）節点と呼ぶ。ATの根をaとし、a'を条件を満足するオカーランスとする。このa'に対して、各t_iの条件を満足するオカーランス集合をT_iとする。t_1，…，t_nがxの合流節点である事は各T_iが同一でなければならない事を意味している。



a）アクセス木（not DFA）



b）DFAによるアクセス木

o：合流ノード

図3.19

我々のDFAによって生成されたATでは、c(x)内のある節点をt_iとすると、t_iを根節点とする部分木内にc(x)の全ての節点が含まれる〔図3.19〕。この性質は重要である。t_iのオカーランスt_i^*からアクセスされるc(x)内の全てのt_i（≠t_i）の各オカーラン

ス t_i に対して、t_i との同一性の判定を加えるだけで、合流点の条件が調べられる〔図3.19のb)の場合〕。

この性質を有しなければ、T_i と T_j の格納の為の中間ファイルと互いの共通部分を求める集合演算が必要になってしまう。従って、我々のDFAでは、中間結果を必要としていないアクセスパスを生成できる。

アクセス木生成の例として、図3.20のCQGに対するアクセス木は、図3.21の様になる。図3.21のJは合流節点になっており、葉の節点(一番下の節点)をアクセスした時に、最初にアクセスしたオカーランスと同一であるかを判定する。

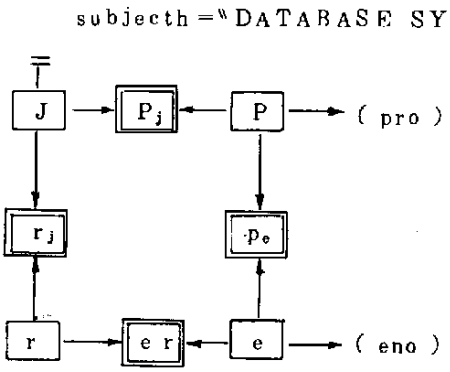


図3.20 CQG

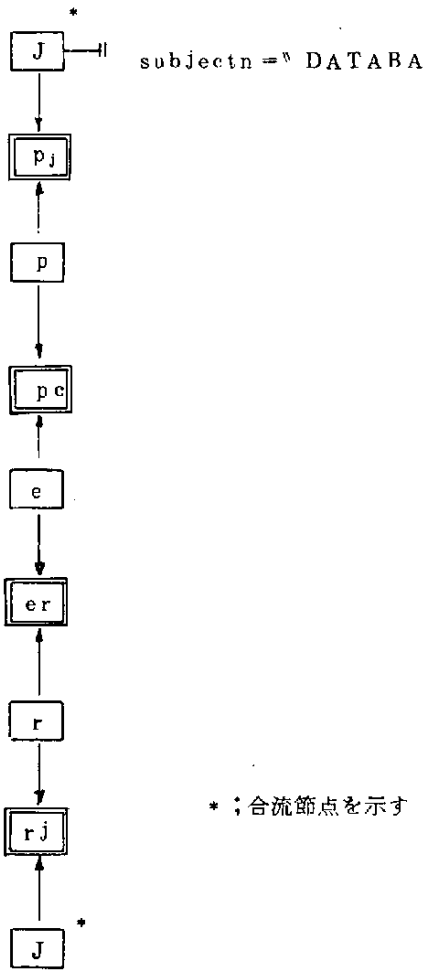


図3.21　図3.20のCQGに対するアクセス木

C、アクセス木（ＡＴ）の定義

アクセス木（ＡＴ）は、オカーランス変数を表わす節点集合ＡＮと、セット辺を表わす枝集合ＡＢとから成る。この木をpre-orderにたどった時の一連のセット型、レコード型の対がアクセス順序（即ち、アクセスパス）を表わす。

各節点ａは、対応するＣＱＧ節点（ＣＮ（ａ）と記す）、制限式ｒｆ（ａ）、目標項目集合ｔｌ（ａ）とから成る。枝（ａ，ｂ）∈ＡＢは、次のセット述語Ｓ（ａ，ｂ）を表わす。

$$S(a,b)=\begin{cases} S(a,b) & \underline{\text{if}}\ a\text{はｂの親である。} \\ S(b,a) & \underline{\text{otherwise}} \end{cases}$$

ここで、Ｓ（ａ，ｂ）は、ａがｂの親のとき、Ｓ（ｂ，ａ）は、ｂがａの親の時、真となる述語である。

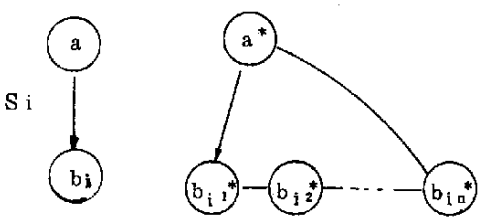
ＡＴ内のａの子節点をchild（ａ）と記す。又、ＡＴ内のある２つの節点間に、ＣＱに対応して、非主辺ｎｊｆ（ａ，ｂ）、不等辺ａ≠ｂを示す辺が存在し得る。

節点ａとセット述語Ｓｉで結合された節点をｂｉとする。各節点ｘの取るオカーランスをx^*と記す。ａのオカーランスa^*に、Ｓｉを介して結合されたｂｉのオカーランス集合をＳｉ(a^*)と記す。条件を満足するとは、次の述語condが真の時である。

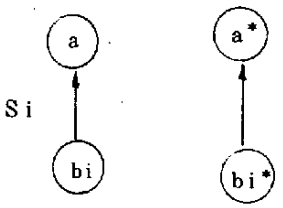
$$\mathrm{cond}(a^*)=\begin{cases} \mathrm{true} & \underline{\text{if}}\ \mathrm{rf}(a^*)\wedge\forall bi\in \mathrm{child}(a) \\ & \quad \exists bi^*\in \underline{Si}(a^*)\ \ \mathrm{cond}(b^*) \\ \mathrm{false} & \underline{\text{otherwise}} \end{cases}$$

即ち、述語condは節点ａが子節点を持つ場合は、全ての子節点が述語condを満足し、自分自身が制限式を満足する時、真となる。

Ｓｉを介して結合されたｂｉのオカーランス集合Ｓｉ（a^*）は、ａがｂｉの親、ｂｉがａの親の場合で図3.2.2ａ）の様になる。



ａ）ａがｂｉの親の場合



ａ）ｂｉがａの親の場合

図3.22　オカーランス集合Ｓｉ（a^*）

図3.2 2のａ）の場合は、述語condはａ* 制限式を満足し、b_{i1}^*, …, b_{in}^* の中の１つでもcondを満足すれば、cond（ａ*）は真となる。ｂ）の場合は、ａ*が制限式を満足し、b_i がcondを満足するとき、cond（ａ*）は真となる。

又、ＡＴ根節点をａとすると〔図3.2 3〕、ａのある一つのオカーランスａ*がcondを満足すれば、問合せは解を持つことになる。

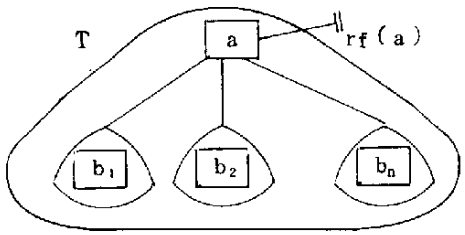


図3.23　アクセス木T

biがセット型Siの子の時、これをM（member）節点、親の時、これをO（wner）節点と呼ぶ。又、ａの子オカーランスの集合Si（a*）は、セット型Siに基づいて全順序付けられている。

CODASYL DMLは、ＣＡＬＣ項目による直接アクセスと、順序集合Si（ a*）のオカーランスをその順にたどる逐次アクセスがある。ここで以下を定義する。

Si（ a*, k ）　＝Si（ a*）内のｋ番目のｂi オカーランス

par（ b_i ）　＝b_i の親節点（＝ａ）

sptr（ b_i ）　＝b_i の右隣りの兄弟節点（＝b_{i+1}）

cptr（ b_i ）　＝b_i の一番左の子節点（＝b_1）

l－next（b_i）＝b_i の先祖の中で、最も近くにいる
M接点（無ければ、ＴＥＲＭ）

この時、各ＡＴ節点における各オカーランスアクセスは、図3.2 4の様になる。

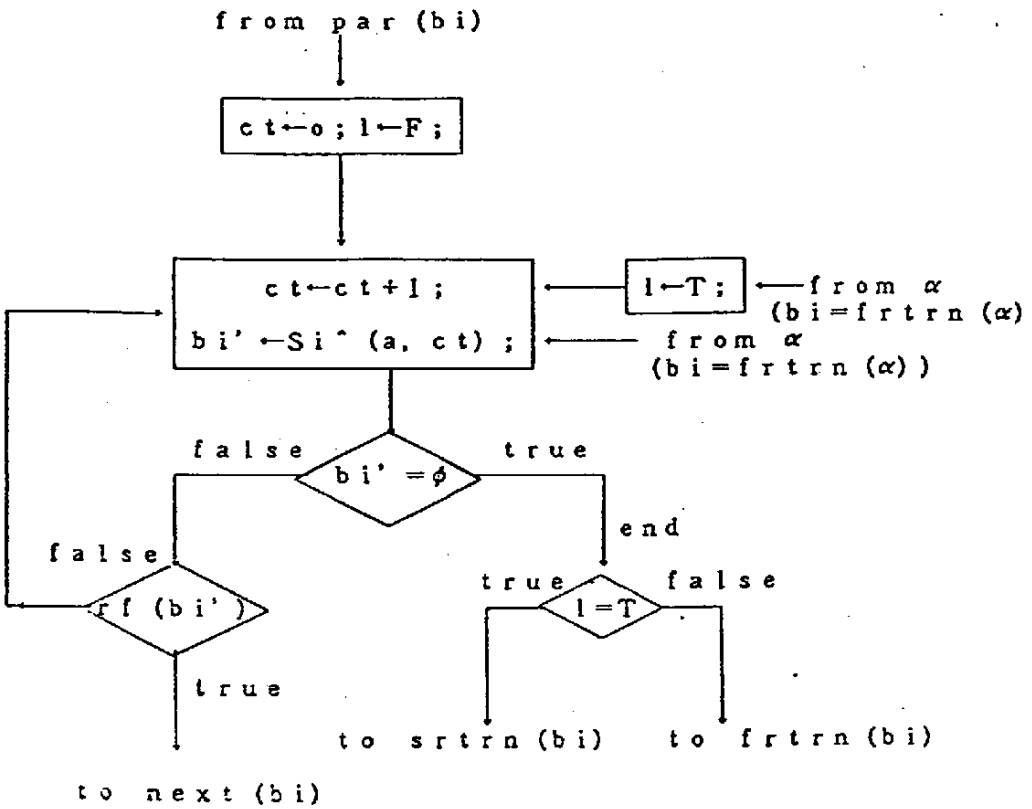


図3.24 AT節点biのDML

図中で、frtrn, srtrn, nextは次の様である。

$$frtrn(b_i) = 1 - next(b_i)$$

$$srtrn(b_i) = \begin{cases} sptr(b_i) & \underline{if} \quad sptr(b_i) \neq \phi \\ 1-next(b_i) & \underline{otherwise} \end{cases}$$

$$next\ (b_i) = \begin{cases} cptr(b_i) & \underline{if} \quad cptr(b_i) \neq \phi \\ 1-next(b_i) & \underline{othenwise} \end{cases}$$

（biは葉節点）

frtrn, srtrn, nextで次にアクセスするオカーランスへの復帰は各々図3.25の様になる。

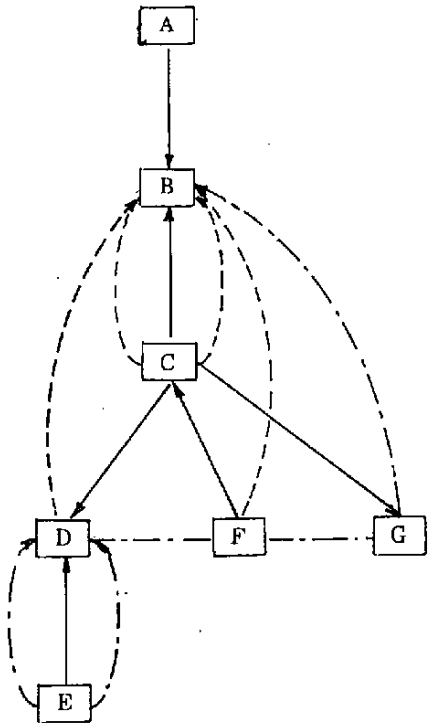


図3.25　成功及び失敗復帰

（ srtrn(b_i), frtrn(b_i) ）

D．アクセスコスト

CODASYL 問合せグラフ（ＣＱＧ）からアクセスパスを生成する為に必要なアクセスコストについて考える。アクセスコストは、目標で述べた様にアクセスされるレコードオカーランス数である。

a）選択度（ selectivity ）

レコード型Xのオカーランス変数をｘとする。ｘの制限式 rf（ｘ）の選択度をｓｔ（ｘ）と記す。これは、存在するｘオカーランスの中で、条件 rf（ｘ）を満足するものの割合の期待値である。表3.2は、rf（ｘ）のパターンと選択度を示している。

図3.2　rf(a)の選択度

rf(a)	選択度 st(rf(a))
r.a＝v	card(a)／card(r) （$=st_{ra}$ と記す）
r.a≠v	$1-st_{ra}$
r.a $\left\{\begin{matrix}>\\<\end{matrix}\right\}$ v	$(1-st_{ra})/2$
r.a $\left\{\begin{matrix}\geq\\\leq\end{matrix}\right\}$ v	$(1+st_{ra})/2$
$r.a=v_1 \vee \cdots \vee r.a=v_n$	$n \cdot st_{ra}$
$rf_1 \wedge \cdots \wedge rf_n$	$st_{f1} \cdot st_{f2} \cdots st_{fn}$ ここで$st_{f1}=rf_1$の選択度
$rf_1 \vee \cdots \vee f_n$	$1-(1-st_{f1}) \cdots (1-st_{fn})$
（rf_iは互いに異なった属性を参照している）	

r.a＝vは、xに対応する組rの属性aが値vに等しいという制限式を表わす。この時の選択度は、属性aの値がvに等しいオカーランスのカーディナリティを全オカーランスのカーディナリティで割ったものとなる。

又r.a＞vは、xに対応する組rの属性aが値vより大きいという制限式を表わす。この時の選択度には、論理的な根拠は無いが、r.a≠vよりは少ないことを表わしている。

ここでは、各属性に対して値が一様に分散していることを前提として選択度を設けている。値が一様に分布していない場合は、データベース毎に経験的に選択度を定めることが有効と考えている。この選択度の妥当性については、評価で検証する。

b）結合度（Connectivity）

セット型S(A, B)に対して、以下を定義する。

$$cnt(S) \triangleq \text{Aに対するBの結合度} \triangleq \sum_{\forall a \in A} |\{b \mid (a,b) \in S\}|_1 / |A|$$

$$icnt(S) \triangleq \text{Bに対するAの結合度} \triangleq \sum_{\forall b \in B} |\{a \mid (a,b) \in S\}|_1 / |B| \triangleq \begin{cases} 1 & \underline{if}\ membership\text{-}class(s)=M/A \\ \leq 1 & \underline{otherwise} \end{cases}$$

（membership-class（S）=M／Aは、Sのメンバシップクラスが MANDATORY　AUTOMATIC であることを表わす。）

ここで、集合Rに対して、|R|はそのカーディナリティを表わすとする。cnt（S）は1つの親オカーランスaに、平均何個の子bが結合されているかを示している。icnt（S）は、子オカーランスbがSの親を持つ確率を与えている。例えば、図3.26の場合、cnt(S)は2，icnt（S）は0.8になる。

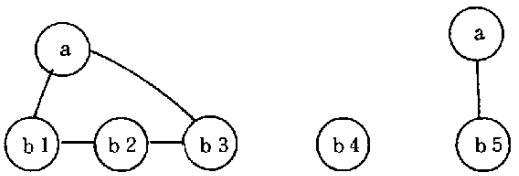


図3.26　S（A，B）の結合度

c）セット型アクセス

アクセス木内では、セット型アクセスとしては、親から子、子から親へのアクセスがある。

まづセット型の親から子へのアクセスを考える。セット型S（A，B）で、Bは制限式rf を有しているとする。

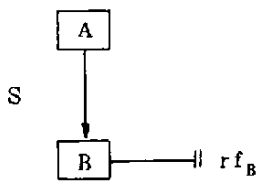


図3.27　セット型S（A，B）

Aのオカーランスaに結合されたSの子の集合をS（a）と記す。S（a）の中から、条件rf_Bを満足する全てのオカーランスを見つける為にアクセスされねばならない平均オカーランス数をacn（S）とする。acn（S）は、rf_B の形式によって、表3.3の様になる。

表3.3　主制限式のパターン

$rf(B) = prf(S) \wedge nrf(S)$

主制限式 prf(S)	条　　件	選　択　度
1) B．a ＝ v	a ＝ s－item(S) ∧ a ≠ D－item(S)	cnt(S) $(1+st_{ba})/2$ st_{ba} は prf(S)の選択度
2) B．a θ v	s－type(S)=STV (S－type(S)＝ST' ∧p_ε ptr $p \varepsilon$ ptr(S))	cnt(S)／2
3) B．a ＝ v	a ＝ D－item(S) a ≠ S－item(S)	cnt(S)／2
4) なし		cnt(S)

θ	ST	ST'
<, ≤	D	A
>, ≥	A	D

1）は、主制限式が等価制限で、属性aがSのソート項目でDNA項目でない場合である。nextポインタをたどって、条件を満足した時点から属性の値が変わった時点でアクセスを打ち切ってよい。従って属性aの選択度を加えた結合度cnt(S)の半分となる。

2）は、制限式がB．a≦v、aがSの昇順ソート項目の時、nextポインタをたどってB．a＞vとなった時点でアクセスを打ち切ってよい。この時、acn(S)は結合度cnt(S)の半分となる。

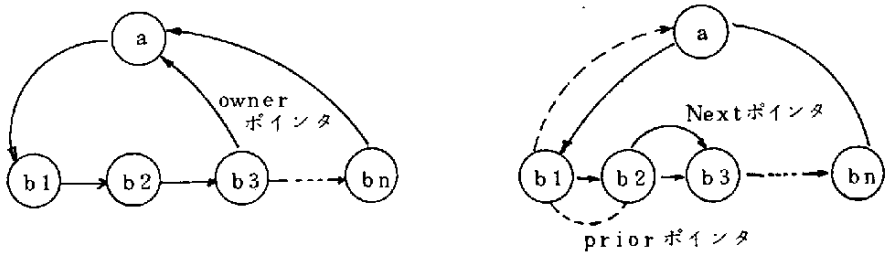
3）は、制限式がB．a＝vで、aがSのDNA項目の時、nextポインタをたどって、B．a＝vが見つかった時点でアクセスを打ち切ってよい。この時、acn(S)は結合度cnt(S)の半分となる。

この様に、アクセスオカーランス数を減少できる制限述語を主制限と呼び、prf(S)と記す。

次に、子から親へのアクセスを考える。iacn(S)をB内のオカーランスbからその親aを見つける為にアクセスされるBの平均オカーランス数とすると以下の様になる。

$$iacn(S) = \begin{cases} 1 & \underline{\text{if}}\ \text{owner ポインタがある。} \\ cnt(S)/2 & \underline{\text{otherwise}} \end{cases}$$

即ち、ownerポインタがあれば、親へのアクセスは1回ですむ。しかし、ownerポインタがなければ、NEXTポインタ又はpriorポインタをたどって親に達しなければならない。この時、子内でアクセスされるオカーランス数の平均をcnt(S)／2とする。



a) ownerポインタがある場合　　b) ownerポインタが無い場合

図3.28　子から親へのアクセス

d) アクセス木(AT)のアクセスコスト

aを根節点とするアクセス木Tを考える.〔図3.29〕。

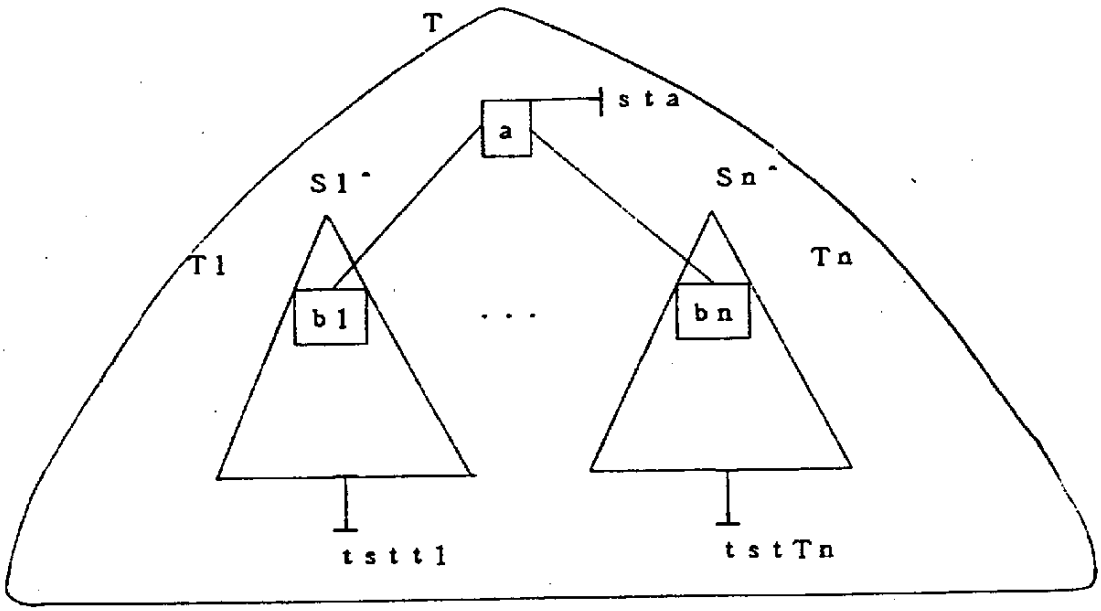


図3.29　アクセス木T

b_1, …, b_nは、aのn個の子でこの順に結合されている。各biは、部分木Tiの根であり、aとはセット型Siによって結合されている。aは、選択度staの制限式を持つ。NOC(T)を、1つのaオカーランスa^*からアクセスされるT内の全オカーランス数の期待値とする。これは、次の様に再帰的に定義される。

$$NOC(T) = 1 + sta \cdot (ct_{ab1} \cdot NOC(T_1) + tst_{T1} \cdot (ct_{ab2} \cdot NOC(T_2) + tst_{Tn} \cdot (ct_{abn} \cdot NOC(T_n)) \cdots\cdots)$$

ここで、

$$tst_T = \begin{cases} st_a \cdot \prod_{j=1}^{n} tst_{Ti} & \underline{if} \quad child(a) \neq \phi \\ st_a & \underline{otherwise} \end{cases}$$

$$ct_{abi} = \begin{cases} acn(si) & \underline{if} \quad biはaの子 \\ iacn(si) & \underline{otherwise} \end{cases}$$

$$st_a = \begin{cases} 1/|A| & \underline{if} \quad aは2番目以降の合流節点 \\ rf(a)の選択度 & \underline{otherwise} \end{cases}$$

tst_T を、木Tの選択度と呼ぶ。これは、Tの根aのオカーランスa が条件を満足する確率である。

C(T)をT内でアクセスされる全オカーランスの期待数とする。Tの根節点aへのアクセス方式としては、ⅰ) CALC項目による直接アクセス、ⅱ) 単項セット型Sによる逐次アクセス、ⅲ) 領域内の逐次アクセスの3種類ある。

ⅰ) の時、aの制限式は次の様である。

$$rf(a) ::= prf(a) \wedge nrf(a)$$

$$prf(a) ::= a.c = v_1 \vee \cdots\cdots \vee a.c = v_n$$

Cは、aのCALC項目で、prf(a)とnrf(a)の選択度を、各々pst_a、nst_aと記す。a内のアクセスされるオカーランス数は、$|A| \cdot pst_a$となる。例えば、CがDNAのCALC項目の場合は、pst_aは1／|A|であるので、a内のアクセスされるオカーランス数は1になる。

ⅱ) では、$ct_{systema}$となる。ⅲ) では、aが全面サーチされるので、|A|個のオカーランスがアクセスされる。

即ち、

$$C(T) = \begin{cases} ⅰ) pst_a \cdot |A| \cdot (1 + nst_{ab1} \cdot NOC(T_1) + tst_{T1} \cdot (ct_{ab2} \cdot NOC(T_2) + \cdots\cdots + tst_{Tn-1} \cdot (ct_{abn} \cdot NOC(T_n) \cdots\cdots) \\ ⅱ) ct_{SYSTEMa} \cdot NOC(T) \quad (SYSTEM経由) \\ ⅲ) |A| \cdot NOC(T) \quad (areaの全面サーチ) \end{cases}$$

となる。

e) 評価関数

b) で述べた、CQG節点xに対する評価関数acen(1, y)について考える。acenとしては、1段階のサーチを行うacc 1と、2段階のサーチを行うacc 2と、3段階のサーチを行うacc 3とがあり、次の様に定義される。

ⅰ) $acc\,1(1, y) = ct_{1xy}$

ii) $acc\,2(l, y) = ct_{lxy} \cdot (1 + st_y \cdot \min_{(m,z)} ct_{myz})$

iii) $acc\,3(l, y) = ct_{lxy} \cdot (1 + st_y \cdot \min_{(m,z)} ct_{myz} \cdot (1 + st_x \cdot \min_{(n,w)} ct_{nzw}))$

但し、yが存在しない時、$ct_{lxy} = 0$と定義する。

ii)、iii)を用いる時、子節点の無いyが優先的に選ばれる（$\because ct_{myz}, ct_{nzw} = 0$）のを防ぐ為に、次の様な方法を用いる。

(1) acc 1(l, y')が最少なy'を選ぶ。

(2) y'が子節点を持つならば（i.e. $ct_{myz} \neq 0$）ならば、子を持つ全てのyの中からacc 2(l, y)(又は、acc 3(l, y))が最少なyを選ぶ。

(3) y'が子を持たなければ、これを選ぶ。

3.5の評価では、i)、ii)とiii)の1段階サーチ、2段階サーチと3段階サーチによるアクセスパス生成を評価する。

3.3.6 COBOL DML 生成

DML生成では、CODASYL 問合せグラフ（CQG）に対して、CODASYL データベースシステム上で実行可能なCOBOL DMLプログラムを生成する。CQGは、CQG分割アルゴリズムにより、連結可能なCQG節点グループ（CQGi）に分割されている。各CQGiに対して手続的なアクセスを表わすアクセス木（ATi）が生成されている。このアクセス木を入力して、COBOL DMLプログラムを生成する。このCOBOL DMLプログラムの実行により中間結果が得られる。これは、非セット辺に対応するアクセスパスがCODASYL モデルに存在しない為である。この過程を図示したのが図3.18である。

DML生成では、次の様にしてCQGに対するCOBOL DMLプログラムを生成する。

1) CQGに対して、COBOL DMLプログラムとして必要な共通部分のCOBOL文を生成する。

```
Identification  Division
        ⋮
Environment     Division
        ⋮
Data            Division
        ⋮               (結果出力の定義 etc)
Procedure       Division
        ⋮               (初期／終了処理 etc)
```

2) 各CQGiに対して、中間結果用ファイルの定義と初期／終了処理を行うCOBOL文を生成する。各CQGiに対する共通部分のCOBOL文を生成する。各CQGiに対して1つ作成する。

3）各ＣＱＧｉに対するアクセス木（ＡＴｉ）を入力してＣＯＢＯＬ　ＤＭＬプログラムを作成する。ここでは、アクセス木（ＡＴｉ）を縦型にたどりながら、各節点に対してパターンマッチングを行ない、必要なＣＯＢＯＬ　ＤＭＬ文を生成する。各節点に対するパターンマッチングは次の様に行う。

① パラメータ（レコード型名、セット型名ｅｔｃ）の設定を行う。

② 節点のポインタ（Ｐ）と親節点のポインタ（ｐｏ）とから図3.30の様にパターンの照合を行う。

③ アクセスパターンに対応するＤＭＬブロックを生成する。

④ ＤＭＬブロック内で使用している項目名を生成する。

⑤ 節点（ｐ）が結果属性を持つ場合、その項目を生成する。但し、節点（ｐ）が合流節点であり、２番目以降に現われたものの場合は生成しない。

⑥ 節点（ｐ）が根節点でＣＡＬＣについての等価制限がある場合は、ＣＡＬＣの値をテーブルに格納するＣＯＢＯＬ　プログラムを生成する。

⑦ ⑤に対応して結果属性を得る手続きを生成する。

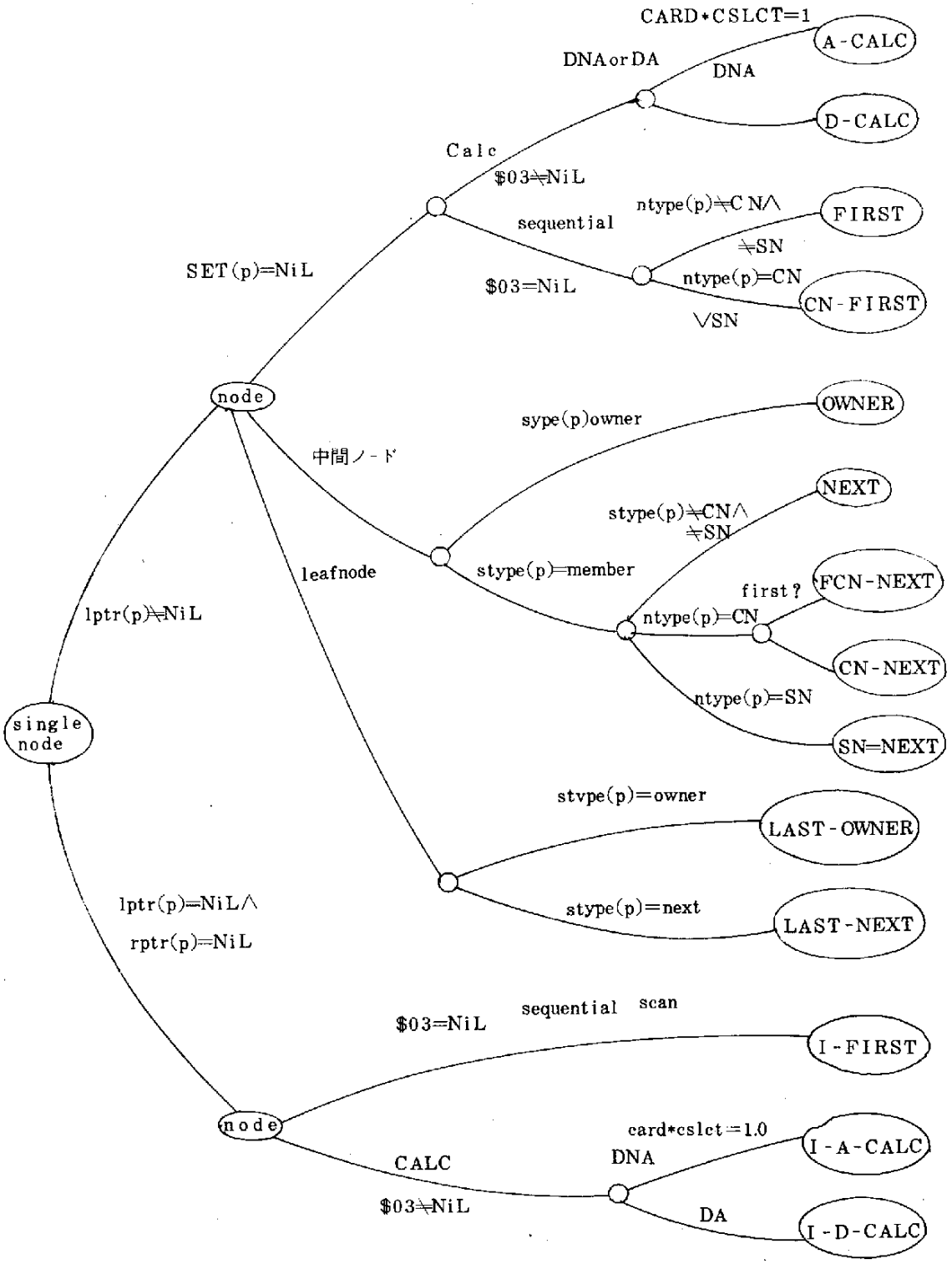


図3.30　アクセスパターンの照会

3.3.7. 出力処理

問合せの結果属性間には，CODASYLスキーマの構造に対して，一般に1対Nの関係が存在している。この為，結果はリレーションとして出力される。結果リレーションは，次の3つのサブルーチンによって生成される。

1) SRTRN

2) FRTRN

3) RESULT

SRTRNは，各DMLブロックからsuccess returnする時にcallされる〔3.3.5 参照〕。更にATにおいて，現在節点より上方の節点にsuccess returoする時（即ち，先祖の節点），working storage(DBK)内のdbキーを用いて結果が出力される。

FRTSNは，各DMLブロックにおいてfailure returnする時にcallされる〔3.3.5 参照〕。この時，DBK内の結果がクリアされる。

RESULTは，条件を満足し，かつ目標属性を有するオカーランスをDBKに出力する時にcallされる。

結果リレーションを作るために，アクセス木から出力木（output tree，OPT）を生成する。出力木を定義する為に，アクセス木に対して次の様な定義を行う。

ATを ATGで生成されたアクセス木

を（AN，ABR）

ここで，　ANをAT節点の集合（節点数をnとする。）

ABRをAT枝の集合

ATの各枝bは，次の様に定義される。

$b(\in ABR)=(a_1, a_2)$

ここで，$a_1, a_2 \in AN$

$parent(b)=a_1$　　a_1は枝bの親節点

$child(b)=a_2$　　a_2は枝bの子節点

πを{1，2，……，n}のあるpermutationとする。$\pi(i)$は，πのi番目の要素とする。例えば，$\pi=\{1, 5, 4, 2, 3\}$の時，$\pi(3)=4$，$\pi(5)=3$である。

この時，$\exists\pi$に対してATNを次の様に定義する。

$ATN \equiv \{ a_{\pi(ti)} \mid i=1, \cdots, n \}$

ここで，$a_{\pi(1)} < a_{\pi(2)} < \cdots\cdots < a_{\pi(n)}$

s.t.

ATをdepth - firstにサーチした時に現われる節点の順番である。

ここで,

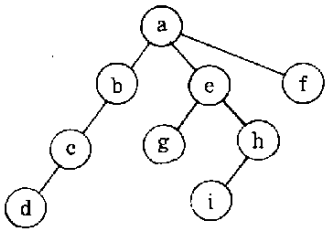
$$\forall a_{\pi(i)} \in ATN \quad A\#(a_{\pi(1)}) \triangleq i$$
$$\triangleq a_{\pi(i)} \text{の順番}$$

とする。

ATに対して，AT節点aとb間のhierarchical path（hpath（a，b））を定義する。

$$\text{hpath}(a, b) \begin{cases} \triangleq \{ a < a_1 < \cdots < a_n < b \} \\ \quad \underline{\text{if}}\ (a, a_1) \in ABR \wedge (a_1, a_2) \in ABR \wedge \cdots \\ \qquad \wedge (a_n, b)\ ABR \\ \triangleq \perp \quad \underline{\text{otherwise}} \end{cases}$$

hpath（a，b）の意味は，あるAT節点aの部分木内にAT節点bが存在する時，このaからbへのパスである。もし，aの部分木内に，bが存在しない時，hpath（a，b）＝⊥（未定義）となる。例えば，図3.3 1のATに対してhpathは次の様に定義される。



hpath(a, d)＝{ a, b, c, d }
hpath(e, i)＝{ e, h, i }
hpath(b, h)＝⊥

図3.3 1　アクセス木

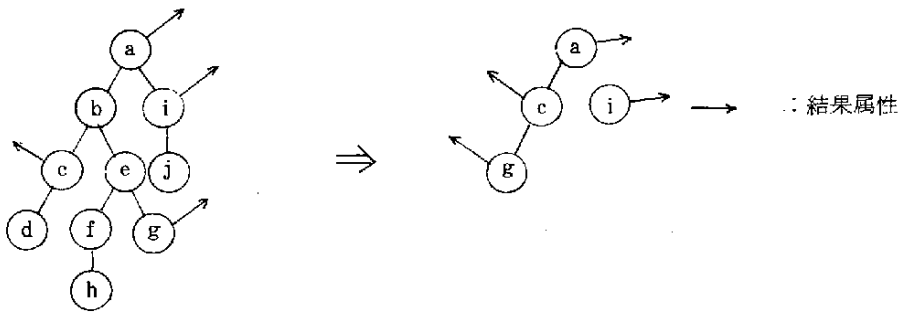
次に出力木（OPT）を定義する。出力木は，AT内の目標属性を有するAT節点を節点とした次の様な木である。

OPT≙（ON, OBR）

ここで

ON≙{ a | a∈AN∧t $l_{(a)} \neq \phi$ }
≙{ a | aは，AT内にあり，かつ目標属性を有したAT節点 }

ob＝（o_1, o_2）∈OBR
（o_1, o_2∈ONかつo_1とo_2との間には，hierarchical pathがある。）
if　$o_1 \in ON \wedge o_2 \in ON \wedge \text{hpath}(o_1, o_2) \neq \perp \wedge$
$\forall o \in \text{hpat}(o_1, o_2) - \{ o_1, o_2 \}, t l_{(0)} = \phi$
（o_1からo_2へのパス上に，結果属性を有するAT節点はない。）



アクセス木（ＡＴ）　　出力木（ＯＰＴ）

図3.3.2　アクセス木から出力木の生成

ＯＰＴを，depth－firstにサーチした時のＡＴ節点の順序集合をＯＰＴＮとする。ここで，ＯＰＴ内のＡＴ節点数をｍとする。また，π_mを｛１，２，……，ｍ｝のあるpermutationとする。

$$OPTN \triangleq \{ o\pi_{m(i)} \mid i = 1, 2, \cdots, m \}$$

ここで，

$$o\pi_{m(1)} < o\pi_{m(2)} < \cdots\cdots < o\pi_{m(m)}$$

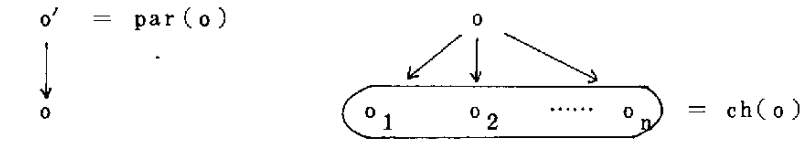
この時，$\forall_{o\pi_{m(i)}}$ ＯＰＴＮ　$o \# (o\pi_{m(i)}) \triangleq i$とする。

ここで，ＯＰＴ内の各節点ｏ（∈ＯＰＴ）に対して

$$par(o) = \begin{cases} o' & \underline{if}\ (o', o) \in OBR \quad OPT\text{のｏの親} \\ \perp & \underline{otherwise} \end{cases}$$

$$ch(o) = \begin{cases} \{ o' \mid (o, o') \in OBR \} \\ \qquad \underline{if}\ (o, o')\ OBR\ OPT\text{内のｏの子集合} \\ \perp \quad \underline{otherwise} \end{cases}$$

これを図示すると次の様になる。



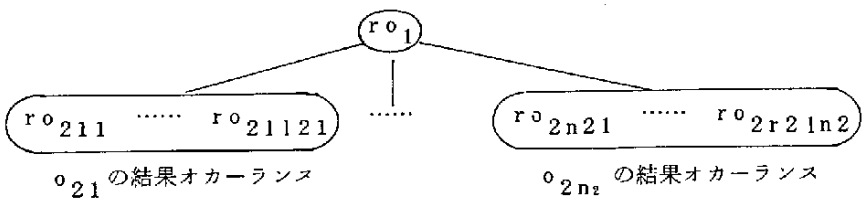
ＯＰＴの意味は，次の様である。ＯＰＴ内の各節点o_1（∈ＯＮ）に対して

$$ch(o_1) \triangleq \{ o_{21}, o_{22}, \cdots\cdots, o_{2n_2} \}$$
$$(o_{21} < o_{22} < \cdots\cdots < o_{2n_2})$$

この時，o_1内のある出力されるオカーランスに対して各o_{2i}は，一般に複数個の出力オカーランスを持つことになる。

$ro_1 \triangleq o_1$のレコードオカーランス

$v_{2i}(ro_i)$ ≙ { ro_{2i} | ro_1 に関連した o_{2i} 内の出力されるレコードオカーランス }
また，$card(v_{2i}(ro_i)) = l_{2i}$ (≥ 1) とする。この時，1つの ro_1 に対して条件を満足するオカーランスの組合せは，$v_{21}(ro_1)$，……，$v_{2n_2}(ro_1)$ の直種となる。



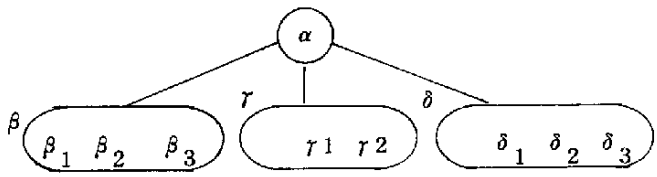
$$v = v_1(ro_1) \times v_{21}(ro_1) \times \cdots\cdots \times v_{2n_2}(ro_1)$$

$$card(v) = card(v_1(ro_1)) \times card(v_{21}(ro_1)) \times \cdots\cdots \times card(v_{2n_2}(ro_1))$$

$$= 1 \times l_1 \times \cdots\cdots \times l_{n2}$$

$$= l_1 \times \cdots\cdots l_{n2}$$

例えば，下図の例を考えると



$v_1(\beta) = \{ \beta_1, \beta_2, \beta_3 \}$，$v_2(\gamma) = \{ \gamma_1, \gamma_2 \}$，$v_3(\delta) = \{ \delta_1, \delta_2, \delta_3 \}$

従って，解 v は，次の様になる。

v					
	α	β_1	γ_1	δ_1	⑥
	α	β_2	γ_1	δ_1	
	α	β_3	γ_1	δ_1	
	α	β_1	γ_2	δ_1	
	α	β_2	γ_2	δ_1	
	α	β_3	γ_2	δ_1	
	α	β_1	γ_1	δ_2	
	α	β_2	γ_1	δ_2	
	α	β_3	γ_1	δ_2	
	α	β_1	γ_2	δ_2	
	α	β_2	γ_2	δ_2	
	α	β_3	γ_2	δ_2	
	α	β_1	γ_1	δ_3	
	α	β_2	γ_1	δ_3	
	α	β_3	γ_1	δ_3	
	α	β_1	γ_2	δ_3	
	α	β_2	γ_2	δ_3	
	α	β_3	γ_2	δ_3	

3.3.8 ビュー定義とアクセス

ビューとは，あるアプリケーションにとって有効なデータ記述である。ビューはお互いに結合可能なビューリレーションの集合から構成される。ユーザは，複数のビューにまたがった利用を行えない。即ち，結合はビュー内の（ビュー）リレーションに対してのみ行なうことができる。

A．ビュー定義

ビューは，ビューリレーションの集合として定義される。ここでvをビューとし，v_1，…，v_nをv内のビューリレーションとする。

また，R_1，……，R_mをvに対するベースリレーションとする。

この時，ビュー定義DEF(v)は，次の様になる。

$$DEF(v)=\{DEF(v_1),\ \cdots\cdots,\ DEF(v_n)\}$$

ここで，$DEF(v_i)=(VAR(v_i),\ def(v_i))\quad (i=1,\ \cdots\cdots,\ n)$

$VAR(v_i)$≜$def(v_i)$が参照するR_1，……，R_mに対する組変数の集合

$def(v_i)$≜v_iの定義式

$$=\underline{define}\ v_i\ (<target-list>)$$
$$\underline{where}<qual>$$

従って，$DEF(V)=\{VAR(v),\ def(v_1),\ \cdots\cdots,\ def(v_n)\}$

$VAR(V)$≜$VAR(v_1)\cup\cdots\cdots\cup VAR(v_n)$

即ち，ビューvは，v_1，……v_nに対して共通の組変数テーブル（RTBL）を持つことになる。これにより，ビューv内では組変数の衝突（同じ変数に対し，異なるリレーションを割り当てる事）を防ぐことができる。

ビューは，defineコマンドによって定義される。

```
define 〔view<v-sehema>〕
       <v-def>〔{:<v-def>}〕;
       <v-def>::=<v-relation>(<targct-list>)
                 where <qual>;
```

ビューvとして，ビュー定義v_1，……，v_nを登録するには，次の様なコマンドで行なわれる。

```
range (x1, x1)……(xm, xm);
define view V
        v1 (tl1) where qual1;          (1)
        v2 (tl2) where qual2;
           ⋮
        vn (tln) where qualn;
```

ここで，$tl_1::=\{va_n=aexp_{11},\ \cdots\cdots,\ va_{1k1}=aexp_{1k1}\}$

$tl_n::=\{va_{n1}=aexp_{n1},\ \cdots\cdots,\ va_{nkn}=aexp_{nkn}\}$

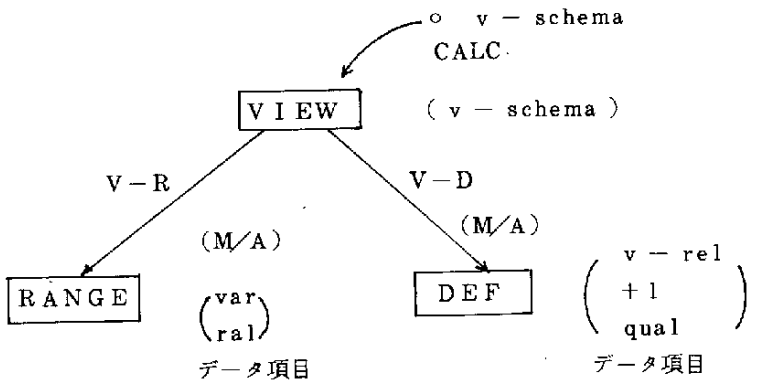
aexpijは，ビュー定義v_iの条件式qualiを満足する組x_1，……x_n上の算術式で，ビュー属性va_{ij}の値を与える。

例として，プロジェクト番号がMMUI80の従業員名と，テーマが「DATABASE SYSTEMS」の論文を書いた従業員名のビューを定義すると次の様になる。

```
RANGE (P,PROJECT) (E,EMPLOYEE) (R,REPORTR) (S,SUBJECT);
RANGE (PE,PROJ-EMP-LNK) (ER,EMP-REP-LNK) (RS,REP-SUBJ);
DEFINE VIEW VIEWTEST
          MMUI-EMP (E.ENAME,P.PNO)
                   WHERE E.@E = PE.@E AND
                         FE.@P =  P.@P AND
                         P.PNO = "MMUI80":
          DBSR-E   (E.ENAME,R.RNO)
                   WHERE R.@R = RS.@R AND
                         RS.@J =  S.@J AND
                          R.@R = RE.@R AND
                         RE.@E =  E.@E AND
                  S.SUBJECTN = "DATABASE SYSTEMS";
```

B．ビュー管理

ビュー管理では，組変数の管理が重要である。そこで，ビューをDBMSで管理する。ビュー管理情報（VMI）は，CODASYLデータベース（AIM）によって管理される。VMIは，下図の様な3つのレコード型（VIEW，RANGE，DEF）と2つのセット型（V—R，V—D）とから成る。

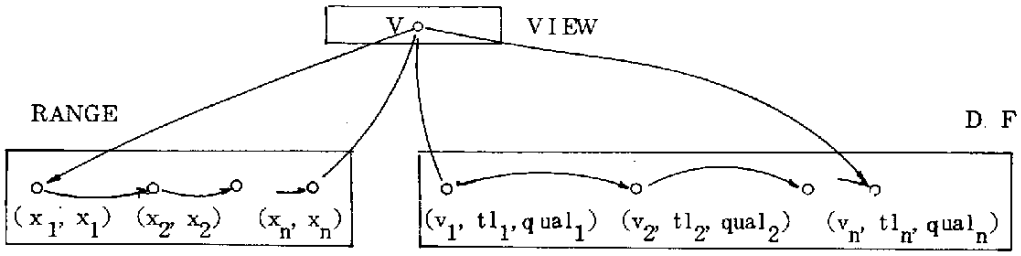


次の様なビューvの定義に対して，ビューデータベースは図の様になる。

range (x_1, X_1)……(X_m, X_m) ;

define view V v_1 ($t1_1$) where qual ;

v_n ($t1_n$) where qual ;



ビューを管理するコマンドには，次の５つがある。

- define コマンド
- go コマンド
- drop コマンド
- destroy コマンド
- load コマンド

a) define コマンド

define コマンドによって定義されたビュー定義文の構文チェックを行い，同一ビュースキーマ名が登録されていなければSTOREする。済に登録されている場合は，STOREされない。但し，define文をeditした後のgoコマンドを入力した場合，新しいビュースキーマに置き換える。

b) go コマンド

define文をeditした後のgoコマンドが入力した時，済に登録されたビュースキーマを削除し，新しいビュースキーマを登録する。

c) drop コマンド

ビュースキーマ内の指定されたビューリレーション定義を削除する。

d) destroy コマンドで指定されたビュースキームを削除する。

e) load コマンド

loadコマンドで指定されたビュー名により，ビュー名，ビューリレーション定義をビュー管理情報からロードする。

C．ビューへのアクセス

ビューリレーションを参照するビュー問合せは，ビューリレーションを対応するＬＣＳリレーションを参照するＬＣＳ何合せに変形することによってアクセスすることができる。これを問合せ変形法（ query modification ）〔 STONM76 〕と呼ぶ。

ビューに対する問合せは，まづ最初に参照しようとするビューがコア内にある場合は，range文とretrieve文によって作成する。コア内に存在しない場合，loadコマンド（ lood＜v－scheme＞；）によってロードを行なう。

load ＜v－scheme＞；　　（ビューがコア内にある場合は，省略）

range （v_1, V_1）……（v_n, V_n）；

retrieve　into　R（$r_1 = ae_1$, ……, $re = ae_k$）

where　＜qual＞

ここでv_iは，ビューリレーションV_iに対する組変数である。Rは，V_1, ……, V_nから生成される結果リレーション名である。r_iはRのi番目の結果属性である。

ae_iは，ビュー属性と定数上に定義された算術式である。qualはビュー属性上に定義される論理式である。

ここで，次の様な置換を導入する。

$\sigma = r/a$

これは，ある文字列aをrで置き換えることを表わしている。

$\sigma\, b = b$

$\sigma\, a = r$

$\sigma(a\beta b) = r\beta\, b$　　但し，β {+, −, *, /, **}

$\sigma(a\theta b) = r\theta\, b$　　但し，θ {<, ≦, >, ≧, =, ≠}

$\sigma(f\theta g) = \sigma\, f\theta\, g$

$\sigma(f\theta g) = \sigma\, f\theta\, g$　　但し，α { ∧, ∨, ～}

（fとgとは，α, β, θを含む式とする。）

これを用いると，ビュー問合せは，次の様になる。

range （v_1', V_1'）……（v_m', V_m'）；

retrieve　into　R′（$r_1' = ac_1$, ……, $r_k' = ae_k$）

where　qual ∧ $qual_1$ ∧ …… ∧ $qual_n$；

この様な置換を有限回繰り返すことで，ビューを参照している属性を全てＬＣＳ属性に置き換えることができる。

D．ビューの実現

ビューを含んだ関係問合せ（ＱＵＥＬ問合せ）を入力してＬＩＳＳＩ（ESR, RSR, ATT）問合せ情報（Ｑ－tree, RRTBL, RTBL, ATTBL）とビュー情報（VIEW−SAVE, VATTBL）を使用して，ＬＣＳリレーションのみを参照するＱ－treeに変換する。

ビュー定義は階層構造をもつことが出来，この階層構造はビュー定義木であらわされる。ビュー定義木を表現するために，関係参照木（ＲＲＴ）を生成する。ビュー問合せをＬＣＳリレーションだけを参照する問合せへ変換する手法としてquery modification 手法を用いる。

木の葉節点が表わすＬＣＳリレーションによって，木内の節点が表わすビューリレーションを下から上へ順々に変換していくことが必要である。

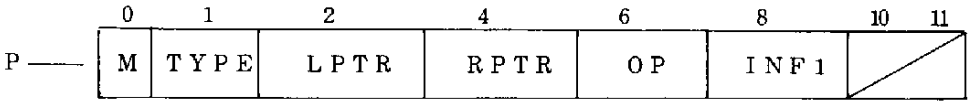
ビュー問合せからＬＣＳリレーションのみを参照する問合せへの変換は，次の３つの処理

から成る。

- 関係参照木の生成
- qery modification処理（ 3.4.3 Bのb. Query modificationの実現を参照）
- Q－trecの正規化

① 関係参照木（ＲＲＴ）の生成

関係参照木は，メモリ内の自由節点空間（ＦＮＳ）につくられる。ＲＲＴの節点の構造は次の様な構造になっている。



ＴＹＰＥ〔Ｐ〕＝ＲＲＴ（＝１１）

ＯＰ〔Ｐ〕＝ＲＲＴＢＬ番号

ＩＮＦ１〔Ｐ〕＝　ＲＴＢＬ番号（問合せの結果属性の場合は，０）

ビュー定義の階層は，一般にｎ－ａｒｙ木構造をしている。例えば，V_1がV_{11}，V_{12}，V_{13}のビューを参照し，V_{11}がV_{111}，V_{112}のビューを参照し，V_{12}がV_{121}のビューを参照し，……，V_{132}がV_{1321}，V_{1322}のビューを参照しているとする。この時，V_1ビュー定義の階層構造をｎ－ａｒｙ木表現で書くと図3.3 3の様になる。

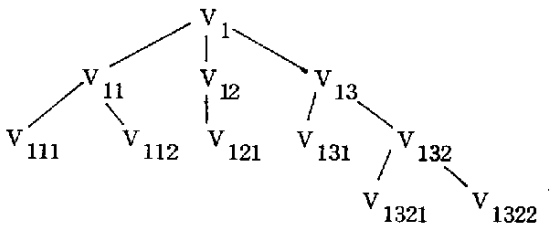


図3.3 3　ｎ－ａｒｙ木

このｎ－ａｒｙ木表現を内部的には自由節点空間に，次の様な２分木表現に直す。

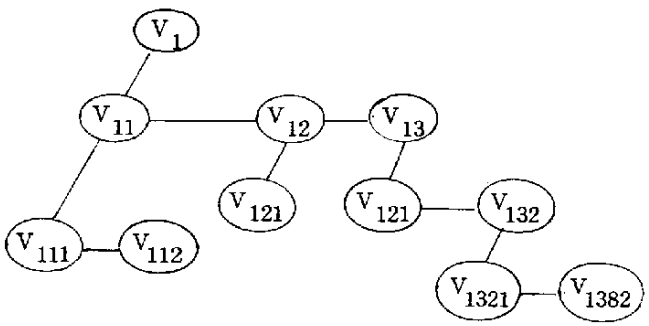


図3.3 4　２分木表現

3.3.9 ＣＯＤＡＳＹＬ　ＤＢＭＳの起動方法

問合せ変換によって生成されたCOBOL DML プログラムをＣＯＤＡＳＹＬデータベースシスシステム上で実行させる為には，プリコンパイル（ＡＤＢＳでは不用），コンパイル及びリンクを行うことが必要である。また，ＣＯＢＯＬ　ＤＭＬプログラムの検索結果をユーザに出力する事が必要である。この為には，プロセス間通信機能が必要である。しかし，メーカが提供するＯＳとしてはプロセス間通信機能を提供していない。

この為，当協会で開発したＪipnetの対話型プロトコル（ＩＴＰ）が提供するプロセス間通信機能を利用したローカル通信処理（ＬＣＰ）が考えられる。ローカル通信処理では，ＩＴＰを用いて，プリコンパイルから実行までのジョブを行う制御文をリモートエントリシステム（ＲＥＳ）に送り，ＣＯＢＯＬ　ＤＭＬプログラムを実行させる。又，検索結果を同様にして，ＩＴＰを経由してＬＤＰ－Ｖ1.5に返される。また，当協会のＭ－１６０がＭ－１７０Ｆにリプレースされ，対話型プログラミングファシリティ（ＩＰＦ）を利用することが出来るようになった。ＩＰＦは，ユーザプログラム（ＰＬ／Ｉ，ＣＯＢＯＬ，ＦＯＲＴＲＡＮ）内からＴＳＳコマンドを実行させることが出来る。この為，ＣＯＢＯＬ　ＤＭＬプログラムのコンパイルし，ＡＩＭ上で実行させ，結果のユーザへの出力をＩＰＦで行うことができる。

Ａ．ローカル通信処理（ＬＣＰ）

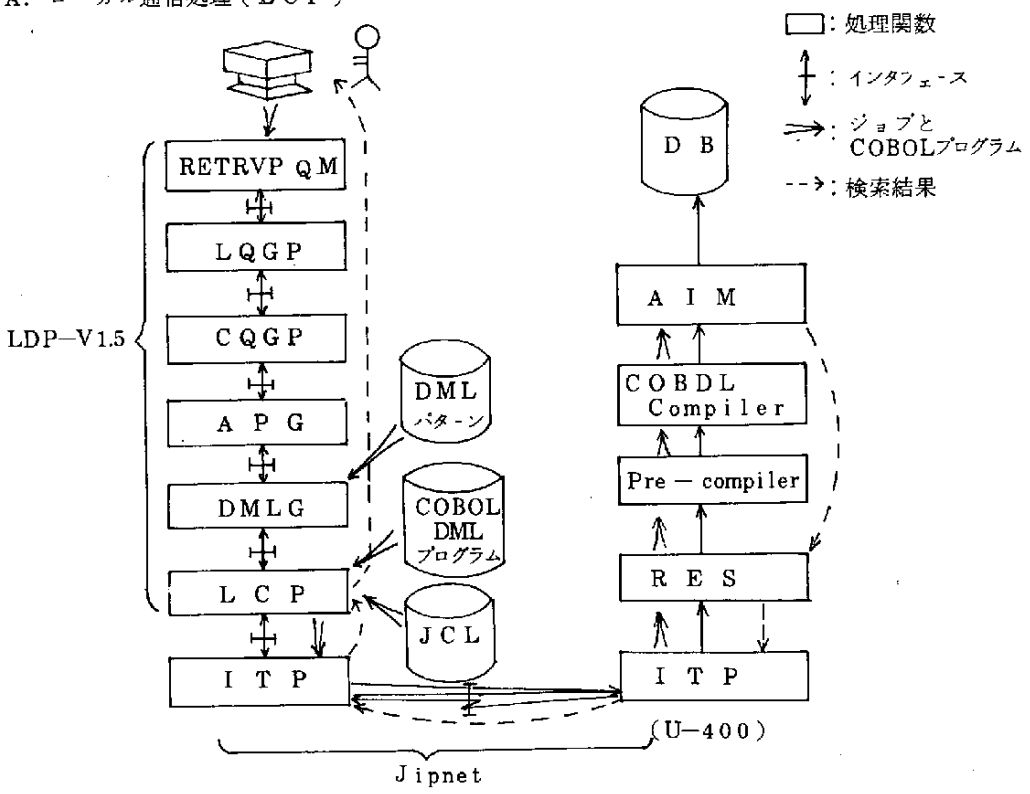


図3.3５　ローカル通信処理の概要

ローカル通信は，ホストとネットワークで結ばれているミニコンピュータU－４００上のＩＴＰと，ユーザプログラムにリンクされたＩＴＰ間で行なわれる。ローカル通信処理を行う為には，次の様に，システムを起動する必要がある。

① ＮＥＴ（ＶＴＡＭ）の起動

② ＴＳＳの起動

③ ＲＥＳの起動

④ ＡＩＭの起動

⑤ Ｕ－４００のシステムの起動

これらのシステムが正常に起動されて始めて，ローカル通信処理を行うことが出来る。ローカール通信処理の処理手順は次の様である。

ローカル通信処理の処理手順

① ＩＴＰプログラムの初期処理

② ＲＥＳとのコネクションを確立

③ ジョブ制御文を送信

④ システムメッセージの受信

⑤ ジョブ結果の受信

⑥ ＲＥＳとのコネクションの解除

⑦ ＩＴＰプログラムの終了処理

これらの処理は，ＩＴＰで提供する次の５つのコマンドによって実行される。

- ＩＴＯＰＮコマンド

  応用プログラム名を引数として受け取り，ＩＴＰの初期設定とＡＣＢのopen処理を行う。応用プログラム名はあらかじめＶＴＡＭに登録しておかなければならない。

- ＩＴＩＮＴコマンド

  ＩＴＯＰＮ，ＩＴＣＬＳ，ＩＴＳＮＤ，ＩＴＲＣＶコマンド要求前の割り込み信号の処理を行う。

- ＩＴＳＮＤコマンド

  コマンド又はメッセージの送信処理を行なう。このコマンドを使用してジョブ制御文が送信される。

- ＩＴＲＣＶコマンド

  メッセージの受信処理を行なう。システムメッセージやジョブ結果はこのコマンドによって受信される　ローカル通信処理では，ジョブ結果だけをユーザに出力する様にする。

- ＩＴＰＣＬＳ

  ＩＴＰの終了処理およびＡＣＢのclose処理を行なう。

これらのコマンドを使用してローカル通信処理（ＬＣＰ）は実現されている。〔３，４，３のｋローカル通信処理の実現を参照〕。

B．ＩＰＦによるインタフェース

ＩＰＦは，ＦＡＣＯＭ　ＯＳⅣ／Ｆ４のもとで使用されるソフトウェアで，応用プログラムでＴＳＳシステムを利用することを可能にしている。ＩＰＦを利用する為には，システムの特別な起動は必要がない。ＴＳＳシステムが起動されていれば良い。

ＩＰＦによるＣＯＢＯＬ　ＤＭＬプログラムの実行及び検索結果の出力は，各々ＴＳＳコマンド“ＳＵＢＭＩＴ”“ＯＵＴＰＵＴ”コマンドを実行することによって実現される。又，実行されたジョブの状態については，“ＳＴＡＴＵＳ”コマンドによって見ることができる。

ＩＰＦを利用したＣＯＢＯＬ　ＤＭＬプログラムの実行は次の図3.3６の様になる。ＩＰＦを利用し，ユーザインタフェースを改良したシステムとしてＲＩＳＣＯＳを検討している〔付記Ⅰ　ＲＩＳＣＯＳを参照〕。

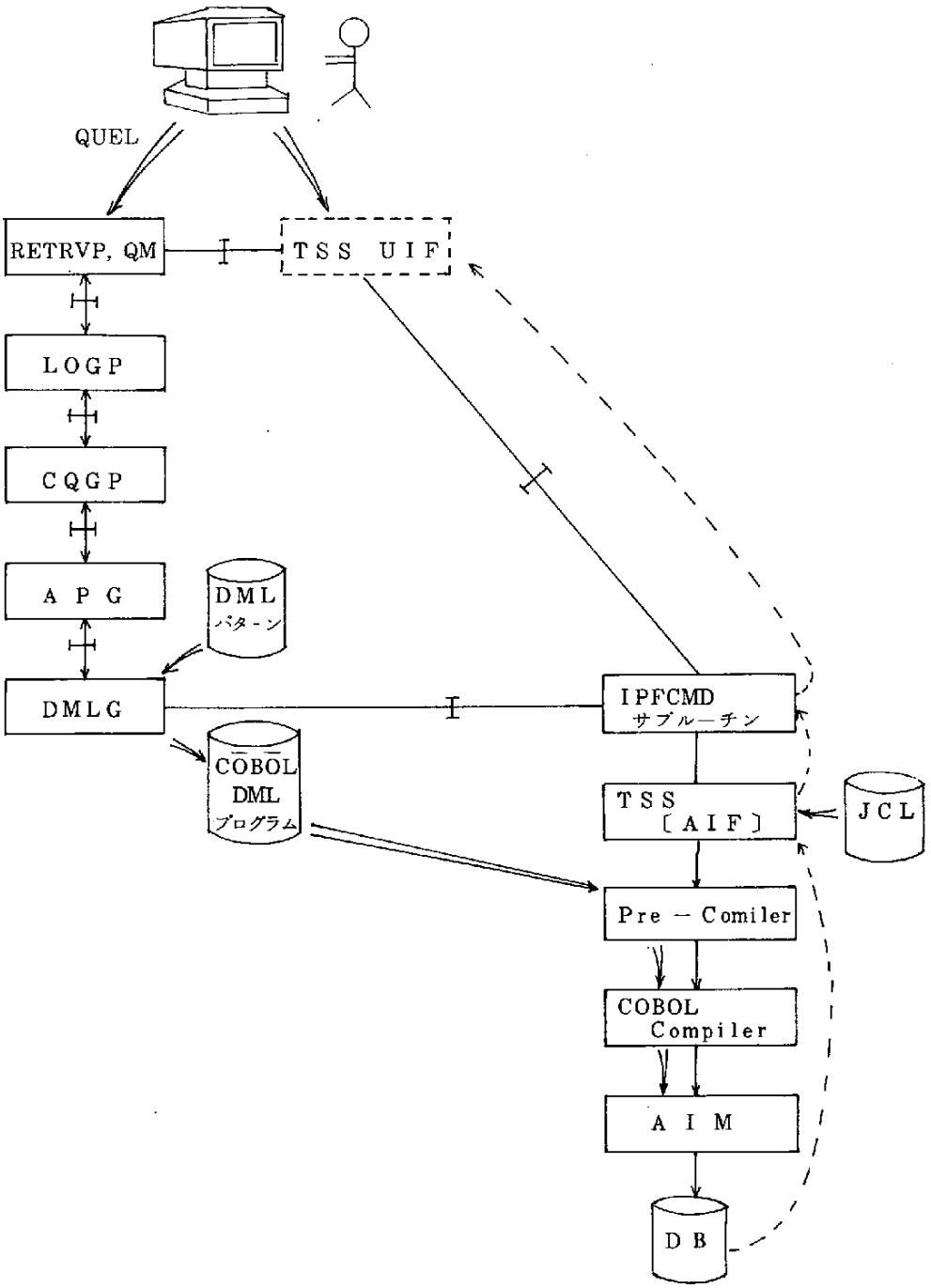


図3.3.6 M－170F IPF version

3.4　ＬＤＰ－Ｖ 1.5の実現

本節では，ＣＯＤＡＳＹＬデータベース上のリレーショナルインターフェイス（ＬＤＰ－Ｖ1.5）の実現について述べる。ＬＤＰ－Ｖ 1.5は，スキーマ変換プロセッサと問合せ変換プロセッサ及び異種性情報から成っている。3.4.1では，システム構成について，3.4.2ではスキーマ変換プロセッサ（ＨＩＰ）について，3.4.3では，問合せ変換プロセッサ（ＱＴＰ）について，3.4.4では，まとめと今後の課題について述べる。

3.4.1　システム構成

ＬＤＰ－Ｖ 1.5は，スキーマ変換プロセッサ（ＨＩＰ），問合せ変換プロセッサ（ＱＴＰ）及び異種性情報（ＨＩ）とから成る。ＬＤＰ－Ｖ 1.5のシステム構成は，図3.3 7の様になっている。

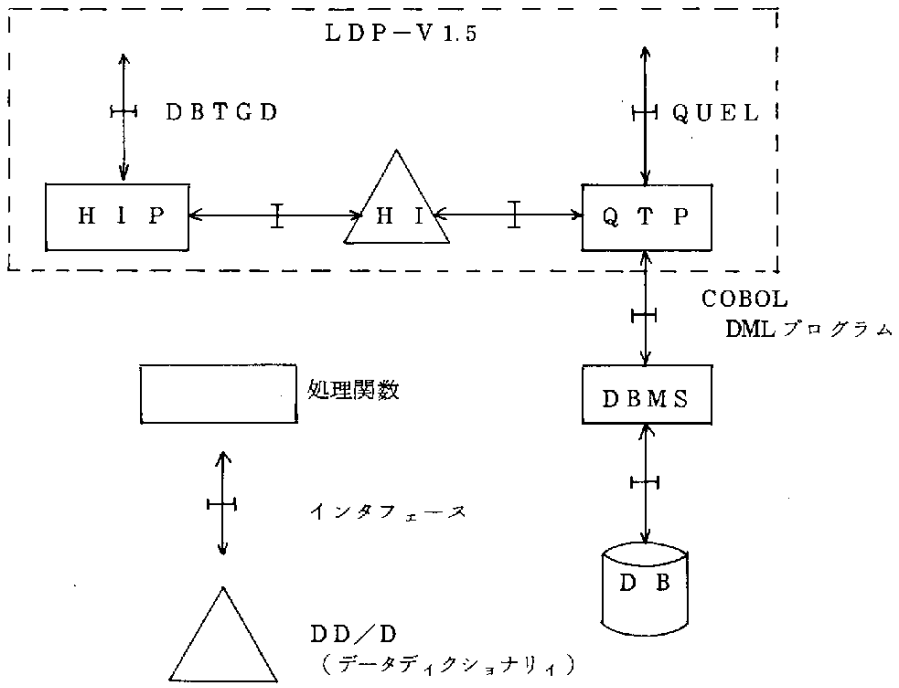


図3.3 7　システム構成図

スキーマ変換プロセッサ（ＨＩＰ）は，ＣＯＤＡＳＹＬスキーマを記述したＤＢＴＧＤを入力して，リレーショナルスキーマ（ＬＣＳ）を生成する。この時に，異種性情報（ＨＩ）としてＬＣＳ，ＬＩＳ，ＬＣＳとＬＩＳの対応情報及びＤＭＬ情報を生成する。ＨＩＰで生成されたＨＩは，問合せ変換プロセッサ（ＱＴＰ）で利用される。

問合せ変換プロセッサ（ＱＴＰ）は，リレーショナル問合せ（ＱＵＥＬ問合せ）を入力し，HI

を参照してＣＯＢＯＬ　ＤＭＬプログラムを生成する。

3.4.2　ＨＩＰ（スキーマ変換プロセッサ）

スキーマ変換プロセッサでは，ＤＢＴＧＤ（ＣＯＤＡＳＹＬスキーマ記述）及びＤＭＬＤ（ＤＭＬ記述）を入力し，ＣＯＤＡＳＹＬスキーマ（ＬＩＳ）からリレーショナルスキーマ（ＬＣＳ）を生成する。スキーマ変換プロセッサの概要は，図３.３８の様になっている。

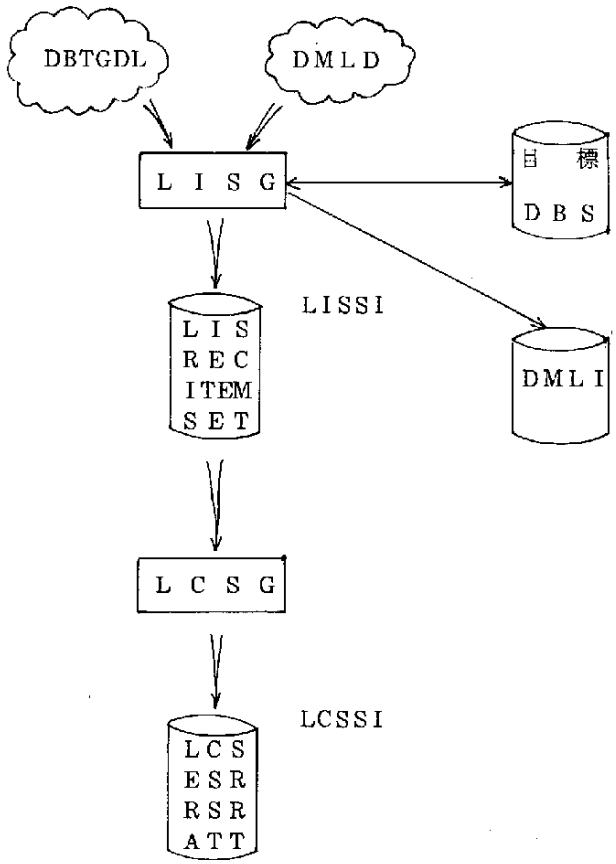


図３.３８　ＨＩＰの処理概要

ＨＩは，次のリレーションから成る。

- ＬＩＳＳＩ　（ＬＩＳ，ＲＥＣ，ＳＥＴ，ＩＴＥＭ）

- LCSSI ⟨ LCS / ESR / RSR / ATT
- DMLI — DML

LISGは，DBTGDとDMLDを入力して，LISSIの4つのリレーション（LIS, REC, SET, ITEM）とDMLIのDMLリレーションとを生成するプログラムである。また，同時にデータベースをアクセスし，LISSIのリレーションにパフォーマンス情報（カーディナリティ，選択度，結合度）をセットする。

LCSGは，こうして得られたLISSIをもとに，LCSSIの4つのリレーション（LCS, ESR, RSR, ATT）を生成する。

A. DBTGD（CODASYLスキーマ記述）

CODASYLスキーマ記述は，CODASYLスキーマ記述言語（DBTGDL）を用いて，CODASYLスキーマのデータ構造を共通記述したものである。即ち，CODASYL モデルの種々のインプリメンテーションとは独立なCODASYLモデルのデータ構造の共通表現が，DBTGDである。

CODASYLモデルのデータ構造は，次の様であると考える。

① レコード型

- 項目の構成
- 各項目についてのアクセスパス

  CALC with DNA — CALC項目で重複が許されない項目。
  CALC with DA — CALC項目で重複が許されている項目。
  non-CALC with DNA — CALC項目以外で重複が許されない項目。
  non-CALC with DA — CALC項目以外で重複が許されている項目。

- リンクレコード型かどうか？

② セット型

- 親レコード型名
- メンバシップクラス

  insertion AUTOMATIC／MANLIAL
  retention FIXED／MANDATORY／OPTIONAL

- 子レコード型名

  項目 { DNA — 重複が許されない。
  DA — 重複が許される。
  { sorted — ソートキーである。
  non－sorted — ソートキーではない。

- 構造制約（structural corstraint）
- sat — selection

CODASYLスキーマ記述言語（DBTGDL）の構文を図3.39に示す。

```
<DBTGDL>::=<DBTG statement>{;<DBTG statement>}
<DBTG statement>::=<record-type-description>|<set-type-description>
<record-type-description>::=REC〔ORD—TYPE〕:<area-name>〔:<rec-mode〕
                                <record-type>
                                    〔(<field — list〕
<rec-mode>::=LINK|NORMAL*
<field-list>::=<field-description>{,<field-description}
<field-description>::=<field name>/<format>
                                〔/<access-mode>〕
<access-mode>::=<calc-mode>/<dna-mode>
<dna-mode>::=DNA/DA*
<calc-mode>::=CALC/NCALC*
<format>::=<type><length>
<type>::=C|D
```

㈱ *は，省略時解釈を示す。

図3.39　DBTGDLの定義(1)

```
<set-type-description>::=SET〔—TYPE〕  <set-type>
                                <owner-deseription>
                                <member-description>
                              〔<structural constraint>〕
<owner-description>::=OWNE〔R〕  <record-type>
<member-description>::=MEMB〔ER〕 <record-type>
                          〔/ <access- mode>〕
<access—mode>::=<dna—att>/<sort-att>〔/<link-mode>
                    〔/<insertion-mode>/<retension-mode>〕〕
<dna-att>::=<attribute-name>|NIL
<sort-att>::=<attribute-name>|NIL
<insertion-mode>::=AUTOMATIC*/MANUAL
<retension-mode>::=FIXED/OPTIONAL/MANDATORY*
<link-mode>::=NEXT|OWNER
<structural-comstraint>::=STRUCTURE(<att-name-of-o-rec>)
                                <att-name-of-m-rec>)
```

図3.40　DBTGDLの定義(2)

例として3.5節の評価で使用したプロジェクト管理データベースのDBTGDは，次の様である。

```
SCHEMA PROJSUB1, PROJDB, TAKIZAWA;
REC : RANGEF
      EMPLOYEE        ( ENO / D5 ,
                        ENAME / C10,              EFNAME / C20,
                        ESEX / C1,                BIRTH-PLACE / C10,
                        BIRTH-YEAR / D2,          BIRTH-MONTH / D02,
                        ALMA-MATER / C30,         MAJOR / C30,   DEGREE / C1,
                        DEG-YEAR / D2,            HIRED-YEAR / D2,
                        RETIRED-YEAR / D2,        DEPT / C20,
                        EPOSITION / C20,
                        TEL / C12,                EXT / D4                );
REC : RANGEA
      PROJECT         ( PNO / C08 / CALC / DNA,
                        PNAME / C100,             PROJ-TYPE / C1,
                        STATUS-F / C03,
                        SYEAR / D2,               EYEAR / D2,
                        BUDGET / D11            );
REC : RANGEE
      REPORTR         ( RNO / C8 / CALC / DNA,
                        TITLE / C200,             SOURCE-N / C100,
                        VOL / C8,                 NUMB / D3,
                        MONTH / D2,               YEAR / D2,
                        FROM-PAGE / D5,           TO-PAGE / D5,
                        TOTAL-PAGE / D5,          NO-OF-AUTHOR / D2,
                        LANGUAGE / C10,           WRITTEN-YEAR / D2,
                        COST / D9,                RTYPE / C1,
                        RSTATE / C1             );
REC : RANGEB
      SPONSUR         ( SNAME / C20,              STYPE / C10             );
REC : RANGED
      SUBJECT         ( SUBJECTN / C100 / CALC / DNA                      );
REC : RANGEC
      PRODUCT         ( PD-NO / C5,               PD-NAME / C30,
                        PTYPE / C1,               COST / D9,
                        YEAR / D2,                LANGUAGE / C10,
                        PSIZE / D5,               OS-NAME / C15           );
REC : RANGEI
      ORGANIZATION    ( ORG-NAME / C50,           ORG-TYPE / C10          );
REC : RANGEG : LINK
      PROJ-EMP-LNK    ( STATUSS / C4,
                        SYEAR / D2,               SMONTH / D2,
                        EYEAR / D2,               EMONTH / D2,
                        PPOSITION / C10,          ROLE / C10,
                        PERCENTAGE / D3         );
REC : RANGEL : LINK
      EMP-REP-LNK     ( AUTH-NO / D2            );
REC : RANGEJ : LINK       PROJ-SUBJ ;
REC : RANGEK : LINK       REP-SUBJ  ;
REC : RANGEH : LINK       PROJ-RE   ;
SET   SPON-PROJ         OWNER     SPONSOR
                        MEMBER    PROJECT    / PNO / PNO / A/ OWNER;
SET   PROJ-PROD         OWNER     PROJECT
                        MEMBER    PRODUCT    / PD-NO / PD-NO / A/OWNER;
SET   P-E               OWNER     PROJECT
            MEMBER      PROJ-EMP-LNK   /    /    /    /  OWNER ;
SET   E-P               OWNER     EMPLOYEE

            MEMBER      PROJ-EMP-LNK   /    /    /    /  OWNER ;
SET   P-S               OWNER     PROJECT
                        MEMBER    PROJ-SUBJ /    / / / OWNER;
SET   S-P               OWNER     SUBJECT
                        MEMBER    PROJ-SUBJ /    / / / OWNER;
SET   S-P               OWNER     SUBJECT
                        MEMBER    REP-SUBJ  /    / / / OWNER;
SET   R-S               OWNER     REPORTR
                        MEMBER    REP-SUBJ  /    / / / OWNER;
SET   R-E               OWNER     REPORTR
                        MEMBER    EMP-REP-LNK/AUTH-NO/AUTH-NO/A/ OWNER;
SET   E-R               OWNER     EMPLOYEE
                        MEMBER    EMP-REP-LNK/ /AUTH-NO /A/ OWNER ;
SET   PRIOR-PROJ        OWNER     PROJECT
                        MEMBER    PROJ-RE   /    / / / OWNER;
SET   POST-PROJ         OWNER     PROJECT
                        MEMBER    PROJ-RE   /    / / / OWNER ;
SET   ORG-EMP           OWNER     ORGANIZATION
                        MEMBER    EMPLOYEE  /    / / / OWNER;
SET   SYS-ORG           OWNER     SYSTEM
                        MEMBER    ORGANIZATION ;
SET   SYS-SPON          OWNER     SYSTEM
                        MEMBER    SPONSOR ;
SET   SYS-EMP           OWNER     SYSTEM
                        MEMBER    EMPLOYEE  /ENO/ENO/A;
```

図 3.4.1

B．DMLD

DML記述（DMLD）は，DMLDLによって，記述される。DMLDは，問合せ変換において用いられるDMLブロックを記述する。この記述は，DMLブロック番号毎に，COBOL DMLの文集合を記述する。

DML　DLの構文を図3.4.2に示す。

<DMLD>::＝<DML-block-def>{<DML-block-def>}

<DML-block-def>::＝<block-no>//<DML-statement>
{//<DML-statement>};

<block-no>::＝P<number>

図3.4.2　DMLDLの定義

C．HI（異種性情報）

HIは，次の様な目的で作られる。

- QTPの全ての処理は，HIだけを用いて行う。
- システムの相違（マシン，データベース，言語）は，HIによって共通記述する。

これにより，マシン，データベース，言語が変ってもHIの変更だけで，QTPの変更は不要である。

この為，HIは次の4種類の情報を保持する。

- LIS（CODASYLスキーマ）の記述情報
- LCS（リレーショナルスキーマ）の記述情報
- LISとLCSの対応情報
- DML情報

また，これらの情報は次の3種類に分割され，各々リレーショナルモデルで，リレーションとして表わされる。

1）LISSI

LISSIは，CODASYLスキーマとシステムの相違を表わした次の様なリレーションから成る。

- LIS　リレーション
- REC　リレーション
- SET　リレーション
- ITEMリレーション

2）LCSSI

LCSSIは，リレーショナルスキーマ情報とLISとの対応情報を表わした次の様なリレーションから成る。

- LCSリレーション

- ＥＳＲリレーション
- ＲＳＲリレーション
- ＡＴＴリレーション

3) ＤＭＬＩ

ＤＭＬＩは，ＱＴＰで参照されるＤＭＬブロック番号と対応するＤＭＬを記述したＤＭＬリレーションを持つ。

ＤＭＬ言語が異なる場合，このＤＭＬリレーションを変更すればよい。

a) ＬＩＳＳＩ

ＬＩＳＳＩは，ＣＯＤＡＳＹＬスキーマをリレーショナルモデルで表わした３つのリレーション（ＲＥＣ，ＳＥＴ，ＩＴＥＭ）と，ＣＯＤＡＳＹＬ型ＤＢＳのスキーマについての情報（スキーマ名，サブスキーマ名etc ）を持つリレーション（ＬＩＳ）とから成る。

各リレーションの構造は，表3.4～3.7の様になっている。

ＬＩＳリレーションは，各データベースについての情報を保持している。ホストコンピュータ名，スキーマ名，サブスキーマ名は，ＤＭＬ生成時に参照される。

ＲＥＣリレーションは，ＣＯＤＡＳＹＬスキーマのレコード型をリレーションで表わしたものである。オカーランス数はアクセス木生成の評価関数によって参照される。又，ロケーションモードやエリア名はＤＭＬ生成で参照される。

ＳＥＴリレーションは，ＣＯＤＡＳＹＬスキーマのセット型をリレーションで表わしたものである。結合度はアクセス木の評価関数によって参照される。

ＩＴＥＭリレーションは，ＣＯＤＡＳＹＬスキーマのレコード型のデータ項目をリレーションで表わしたものである。選択度は，アクセス木生成で評価関数によって参照される。又，INDEX-MODE，UNIQUNESS，TYPE，LENGTH等はＤＭＬ生成で参照される。

表3.4　ＬＩＳリレーションの構成

属　性　名	属性番号	ＲＯＬＥ	ＴＹＰＥ	LENGTH	説　　明
cs-name	1	K	C	16	schema名（ＤＢ名）
lcs-name	2	K	C	16	サブスキーマ名（ＬＩＳの名前）
la-name	3	N	C	16	local administrator 名
host-computer	4	N	C	16	ホストコンピュータ名
os-name	5	N	C	16	ＯＳ名
c-year	6	N	D	1	ＬＩＳの生成年
c-month	7	N	D	1	〃　月

属　性　名	属性番号	ROLE	TYPE	LENGTH	説　　明
c-day	8	N	D	1	LISの生成日
lis-model	9	N	C	4	"REL" =relational "DBTG" =CODASYL DBTG "INS" =IMS
languagc	10	N	C	16	アクセス言語名

表3.5　RECリレーションの構成

属　性　名	属性番号	ROLE	TYPE	LENGTH	説　　明
REC−TYPE	1	K	C	16	record-type
REC−MODE	2	N	D	1	normal(=0) link　(=1)
DEGREE	3	N	D	2	このrecord-typeのデータ項目の数
CARDINALITY	4	N	D	6	オカーレンス数
LOC−MODE	5	N	D	1	CALC　(=1) NON CALC(=2)
AREA−NAME	6	N	C	16	エリア名
MARK	7	N	D	1	OFF(=0) ON　(=1)
WORK	8	N	D	2	

表3.6　SETリレーションの構成

属　性　名	属性番号	ROLE	TYPE	LENGTH	説　　明
set-type	1	K	C	16	セット型名
o-rec-type	2	N	C	16	owner record-type
m-rec-type	3	N	C	16	member record-type
connectivity	4	N	D	7	orec → mrecの平均 connectivity

属　性　名	属性番号	ROLE	TYPE	LENGTH	説　　明
max-connect	5	N	D	7	orec → mrecの最大 connectivity
min-connect	6	N	D	7	orec → mrecの最小　〃
i-connect	7	N	D	7	mrec → orecの平均　〃 (≦1)
sort-type	8	N	D	1	N(=0):not sorted A(=1):acending D(=2):descending
sort-att#	9	N	D	2	ソートキーのatt-no
unique-att#	10	N	D	2	DNA指定のあるatt-no
link-mode	11	N	D	1	NEXT(=0) NEXT+OWNER(=1) NEXT+PRIOR(=2) NEXT+PRIOR+OWNER(=3)
insertion	12	N	D	1	AUTOMATIC(=0) MANUAL　(=1)
relension	13	N	D	1	MANDATORY(=0) OPTIONAL (=1) FIXED　(=2)
structure-o-att#	14	N	D	2	structural constraintのowner att-no
structure-m-att#	15	N	D	2	〃のmember att-no
mark	16	N	D	1	

表3.7　ITEMリレーションの構成

属　性　名	属性番号	ROLE	TYPE	LENGTH	説　　明
REC−TYPE	1	K	C	16	record-type
ATT−NO	2	K	D	2	item番号
ATT−NAME	3	N	C	16	item名

属　性　名	属性番号	ROLE	TYPE	LENGTH	説　　明
INDEX−MODE	4	N	D	1	CALC（＝1） NCALC（＝0）
UNIQUENESS	5	N	D	1	DNA（＝1） DA（＝0）
SELECTIVITY	6	N	D	6	selectivity×1000の値が入る
TYPE	7	N	C	1	"C"：文字 "D"：整数
LENGTH	8	N	D	6	バイト長
DOMAIN	9	N	C	16	定義域名
MARK	10	N	D	1	OFF（＝0） ON（＝1）

b) LCSSI

LCSSIは，リレーショナルスキーマ情報と，CODASYLモデル要素への対応情報を表わしている。LCSSIは，リレーショナルスキーマ情報を表わす3つのリレーション（RSR，ESR，ATT）とリレーショナルスキーマについての情報（アクセス言語，LISとの対応情報）をもつリレーション（LCS）とから成る。

各々のリレーションの構造は，次の表3.8～3.11の様になっている。

表3.8　ESRリレーションの構成

属　性　名	属性番号	ROLE	TYPE	LENGTH	説　　明
ES−REL	1	K	C	16	entity set
DEGREE	2	N	D	2	entity setの属性の数
TARGET−NAME	3	N	C	16	対応する目標LIS要素名（record-type）
TARGET−TYPE	4	N	D	1	LIS要素のタイプ RECORD（＝0） SET　（＝1）

ESRリレーションは，CODASYLスキーマのレコード型に対応するリレーションである。LIS要素（レコード型名）との対応を表すTARGET−NAMEは，構造変換で参照される。又，LCS問合せの入力に対して，リレーションの存在の確認の為に参照され

る。

ＲＳＲリレーションは，ＣＯＤＡＳＹＬスキーマのセット型やリンクレコード型に対応するリレーションである。ＣＯＤＡＳＹＬスキーマの対応するセット型やリンクレコード型の情報（ target-name， target-type， role ）を持ち，構造変換で参照される。又，ＬＣＳ問合せの入力で，リレーションの存在を確認する為に参照される。

ＡＴＴリレーションは，ＣＯＤＡＳＹＬスキーマのデータ項目に対応するリレーションである。データ項目との対応情報（ target-no， tauget-name ）を持ち，構造変換で参照される。又，ＬＣＳ問合せの入力で，リレーションの属性の存在を確認する為に参照される。

ＬＣＳリレーションは，ＬＩＳリレーションとの対応情報を保持しており，構造変換で使用される。

表3.9　ＲＳＲリレーションの構成

属　性　名	属性番号	ＲＯＬＥ	ＴＹＰＥ	LENGTH	説　　明
rs-rel	1	K	C	16	relationship relation名
rs-type	2	N	D	1	T（＝1）　M／A P（＝2）　O／M G（＝3）　linkレコード型
es-no	3	K	D	2	関連づけられたes-relの番号
es-rel	4	N	C	16	〃　es-rel名
target-name	5	N	C	16	rs-relに対応したＬＩＳ要素名
target-type	6	N	D	1	set（＝2） link - record（＝1）
degree	7	N	D	1	属性数
role	8	N	C	16	rs-relがlink-record-typeのとき関連するセット型名 rs-relがset-typeのとき“owner”or“member”

表3.10　ATTリレーションの構成

属性名	属性番号	ROLE	TYPE	LENGTH	説明
REL-NAME	1	K	C	16	entity set or relation-ship set
REL-TYPE	2	K	D	1	ESR(=0) RSR(=1)
ATT-NO	3	K	D	2	属性番号
ATT-NAME	4	N	C	16	属性名
ES-NO	5	N	D	1	rel-typeがRSRのときこのattが属するesのes-no RSRでないとき　0
DOMAIN	6	N	C	16	定義域名
TYPE	7	N	C	1	C：文字 D：整数
LENGTH	8	N	D	6	バイト長
TARGET-NO	9	N	D	2	対応する項目の属性番号
TARGET-NAME	10	N	C	16	対応するrecord-type
ROLE	11	N	D	1	1：key 0：non-kev
mark	12	N	D	1	

表3.11　LCSリレーションの構成

属性名	属性番号	ROLE	TYPE	LENGTH	説明
lcs-name	1	K	C	16	LCS名
la-name	2	N	C	16	local administrator 名
lis-name	3	N	C	16	対応するLIS名
model	4	N	C	4	LCSモデル
language	5	N	C	16	LCSアクセス言語

属　性　名	属性番号	ROLE	TYPE	LENGTH	説　　明
c-year	6	N	D	1	生成年
c-month	7	N	D	1	生成月
c-day	8	N	D	1	生成月

c) DMLI情報

DMLI情報は，問合せ変換において用いられるDMLブロック番号と対応するDMLを記述したDMLリレーションから成っている。DMLIの構造は，表3.12の様になっている。

表3.12　DMLリレーション

属　性　名	属性番号	ROLE	TYPE	LENGTH	説　　明
block-no	1	K	C	4	ブロック番号
pflag	2	N	D	1	0＝¥，%を含まない 1＝　〃　を含む
statement-no	3	N	D	3	ブロック内の文番号
DNL—statement	4	N	C	72	statement

アクセス木の各節点のアクセスパターンのブロック番号〔3.3.6 DML生成を参照〕とのマッチングで参照される。pflagは，PML—statement 内に変換されるパラメータ（レコード型名，セット型名等）を含むか否かを示している。

D．スキーマ変換プロセッサ（HIP）

HIPは，DBTGDとDMLDを入力して，QTPで参照されるHIを生成するプロセッサである。HIPは，まずDBTGDとDMLDを入力して，CODASYLスキーマをリレーショナルモデルで記述したLISSIとDMLIを生成する。次に，LISSIを入力して，CODASYLデータ構造をリレーショナル形式で表現したLCSSを生成する。

HIPのシステム構成は，図3.43の様になっている。

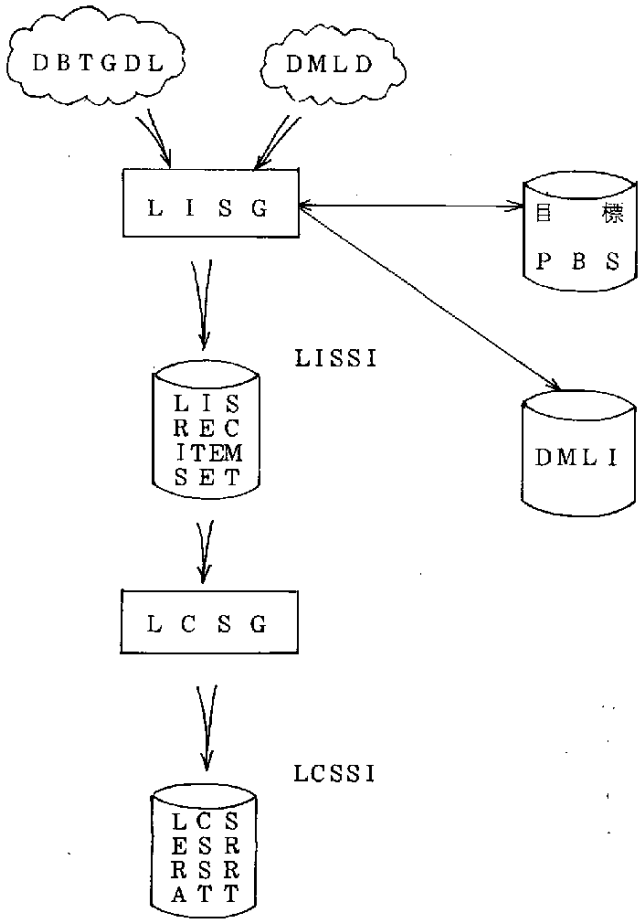


図3.4 3　ＨＩＰの処理概要

a) ＬＩＳＧ

ＬＩＳＧは，ＤＢＴＧＤとＤＭＬＤを入力して，ＬＩＳＳＩの４つのリレーション(LIS, REC, SET, ＩＴＥＭ)とＤＭＬリレーションとを生成する。

また，ＬＩＳＳＩ内のパフォーマンス情報は，目標とするＤＢをアクセスすることによって得る。

ＬＩＳＧの機能は，次の３つから成る。

- ＤＢＴＧからＬＩＳＳＩの生成する。
- ＤＢをアクセスして，ＬＩＳＳＩ内のパフォーマンス情報を求める。
- ＤＭＬＤからＤＭＬリレーションを生成する。

ＤＢＴＧＤからＬＩＳＳＩの生成

ここでは，ＤＢＴＧＤの文字列を読み込み，ＬＩＳＳＩの４つのリレーションを生成す

る。この時，ＲＥＣＯＲＤ文が全て読み込まれた後に，ＳＥＴ文が読み込まれるとする。

処理の概要は，次の様になる。

イ）ＲＥＣＯＲＤ文（ＲＥＣ……；）を読み込む。

ロ）このレコード型の項目名を順にＩＴＥＭリレーションに格納する。

ハ）このレコード型をＲＥＣリレーションに格納する。

ニ）全てのＲＥＣＯＲＤ文の処理が終ったら，ＳＥＴ文（ＳＥＴ……；）を読み込む。

ホ）セット型をＳＥＴリレーションに格納する。

ヘ）ＳＣＨＥＭＡ文（ＳＣＨＥＭＡ……；）からＬＩＳリレーションを生成する。

② ＬＩＳＳＩのパフォーマンス情報のセット

ＬＩＳＳＩのパフォーマンス情報は，実際のデータベースをアクセスすることによって得られる。ここで，パフォーマンス情報とはＲＥＣリレーションのカーディナリティ，ＳＥＴリレーションの結合度，最大，最小，逆結合度（子から親への結合度），ＩＴＥＭリレーションの選択度を指す。

③ ＤＭＬＤからＤＭＬＩの生成

block-noを全てのstatementに附加しながら，ＤＭＬリレーションに出力する。

b) ＬＩＳＳＩからＬＣＳＳＩの生成

ＬＩＳＳＩは，ＣＯＤＡＳＹＬモデル構造をリレーショナル形式で表現したものである。ＬＩＳＳＩを入力として，ＬＣＳＳＩとして4つのリレーション（ＬＣＳ，ＲＳＲ，ＥＳＲ，ＡＴＴ）を生成する。

次の様な処理で，ＬＩＳＳからＬＣＳＳＩを生成する。

イ）ＬＩＳＳＩをサーチして，レコード型Ｒ（≠リンクレコード型）に対してＥＳＲを生成して，Ｒに対する主キー@Ｒを生成する。

ロ）ＬＩＳＳＩは，レコード型を節点，セット型を辺とする有向グラフとして表わされる。ＬＩＳＳＩのグラフのマークを縦型にサーチする。

ハ）辺（セット型）の両端の節点（レコード型）に対してリンクレコード型ならばＧリレーション（ＲＳＲ）を生成する。

ニ）辺に対して，Ｔ又はＰーリレーション（ＲＳＲ）を生成する。

3.4.3 ＱＴＰ（問合せ変換プロセッサ）の実現

問合せ変換では，リレーショナル問合せ（ＱＵＥＬ問合せ）を入力して，ＨＩを参照してＣＯＢＯＬ ＤＭＬプログラムを生成する。生成されたＣＯＢＯＬ ＤＭＬプログラムを実行させ，検索結果をユーザに出力する。

Ａ．ＱＴＰシステム

ＱＴＰシステムは，当協会のＭ－１６０（現在は，Ｍ－１７０Ｆ）上に実現されている。ＯＴＰシステムは，ＣＯＢＯＬ ＤＭＬプログラムの実行，検索結果の出力の相違から次の３つ

のversionがある。

1) M－１７０F　ＩＴＰ

2) M－１７０F　ＩＰＦ

3) ＡＣＯＳ－７００　ＩＴＰ

(1) M－１７０F　ＩＴＰ　version

M－１７０F　ＩＴＰ　version のＱＴＰを構成するモヂュールは図3.4４の様になっている。M－１７０F　ＩＴＰ　version は，ＩＴＰを利用し，ＡＩＭ上のデータベースを検索するシステムである。

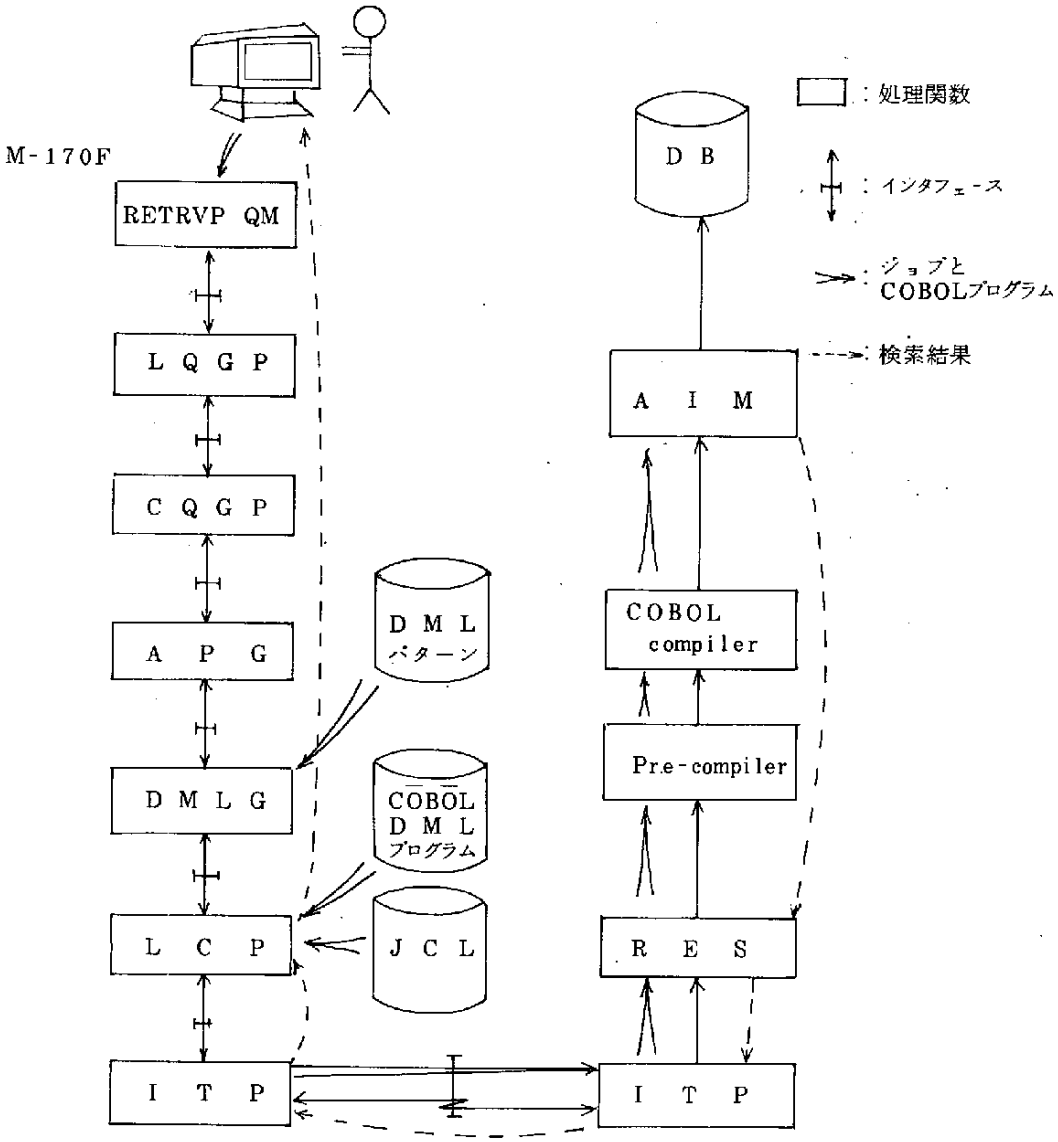


図3.4４　M－１７０F　ＩＴＰ　version

(2) M－１７０Ｆ　ＩＰＦ　version

M－１７０Ｆ　ＩＰＦ　version のＱＴＰを構成するモヂュールは図3.4５の様になっている。コマンドにより，ＩＰＦを利用して，ＴＳＳコマンドを実行させるシステムである。

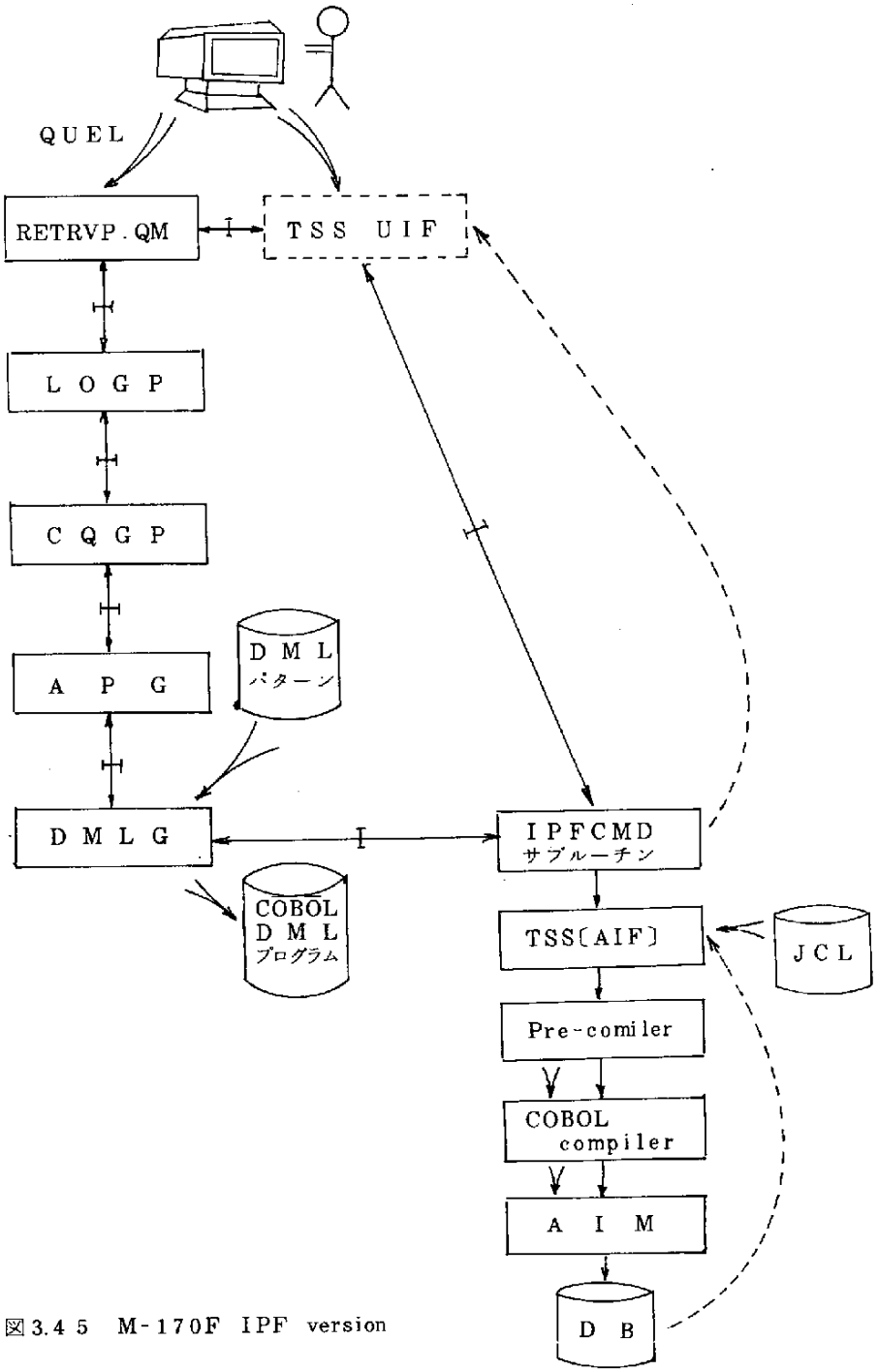


図3.4５　M-170F IPF version

(3) ＡＣＯＳ－７００　ＩＴＰ　version

ＡＣＯＳ－７００　ＩＴＰ　version のＱＴＰを構成するモヂュールは，図3.4６の様になっている。Ｍ－１７０ＦでＡＤＢＳ用ＤＭＬ情報を用い，ＣＯＢＯＬＤＭＬプログラムを生成し，ＩＴＰを使用してＡＤＢＳ上のデータベースを検索するシステムである。ＱＴＰをＡＣＯＳ－７００上で実現していないのは，ＡＣＯＳ－７００ではＰＬ／Ｉ文数が約1,５００程度までしかコンパイルできない（ＱＴＰは，ＰＬ／Ｉ文数約10,000）為である。

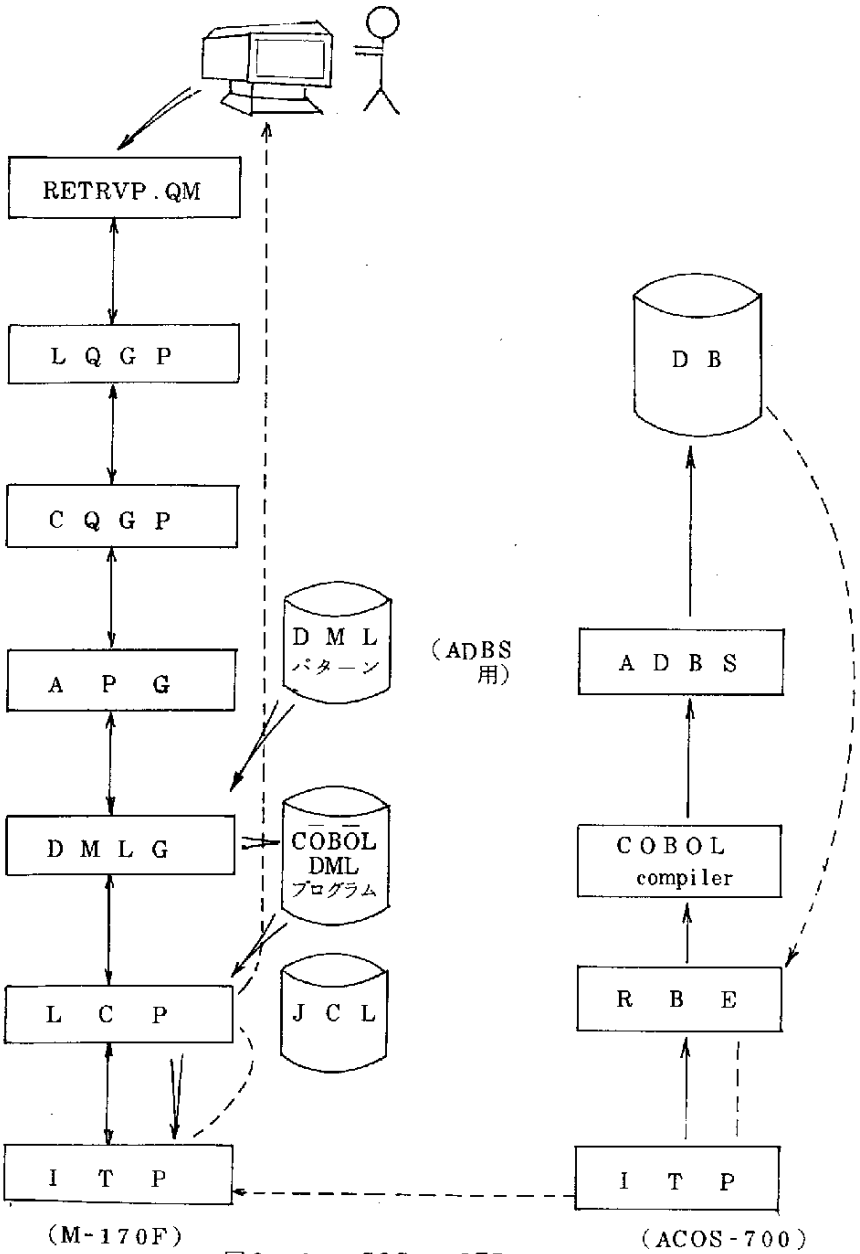


図3.4６　ACOS700 ITP version

B．モヂュールの実現方式

問合せ変換では，ＨＩをコアに常駐化させ，各モヂュールが参照する方式をとった。

a）ＲＥＴＲＶＰの実現

ＱＵＥＬ問合せは，内部表現として２分木（Ｑ－ｔｒｅｅ）と３つのテーブル（ＲＲＴＢＬ，ＲＴＢＬ，ＲＡＴＢＬ）に変換される。この変換を行うプログラムをＲＥＴＲＶＰと呼ぶ。ＲＥＴＲＶＰの処理概要は図3.4 7の様である。

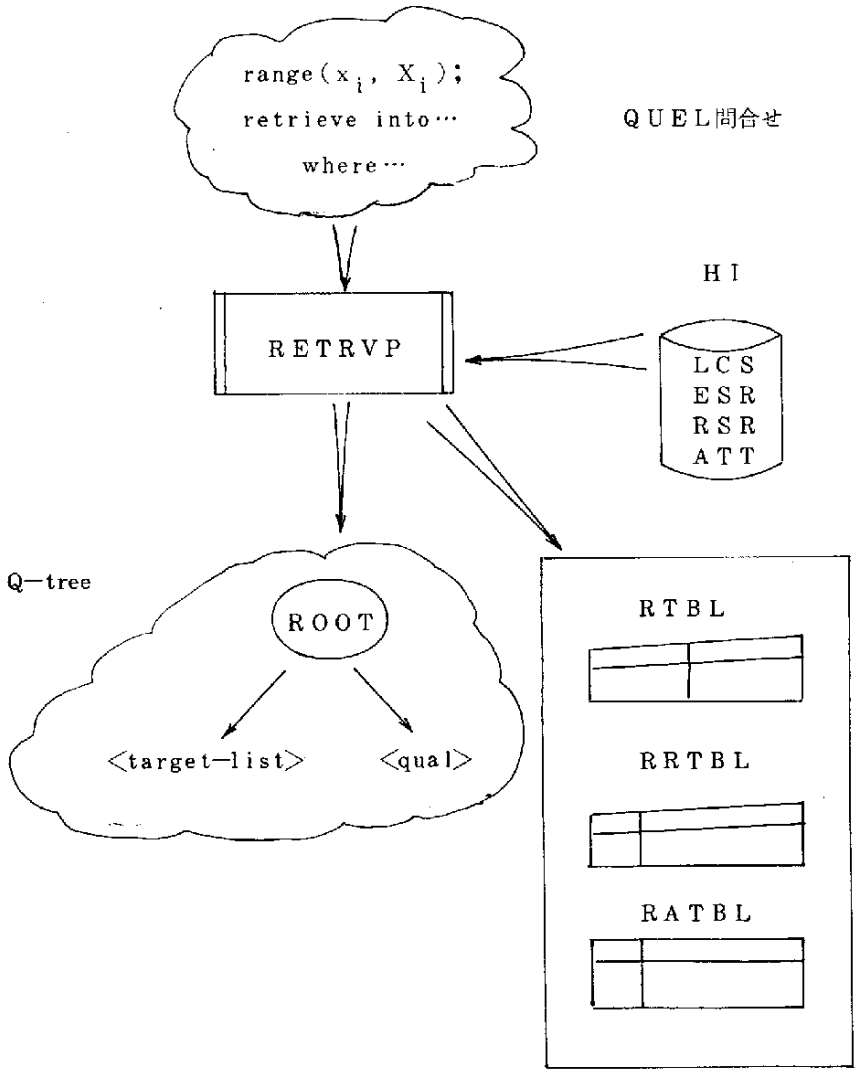


図3.4 7　ＲＥＴＲＶＰの処理概要

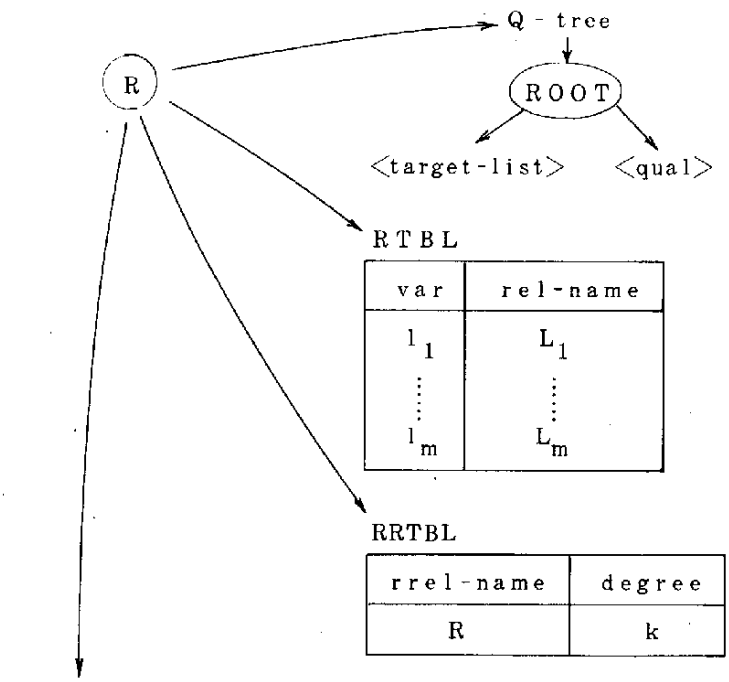
例えば，問合せRは，次の様に表わされる。

問合せR：range　(l_1, L_1)……(l_m, L_m)；

retrieve into R ($a_1 = ae_1$, ……$a_k = ae_k$)

where qual；

内部表現



rrel-rame	att-ro	att-name	role	type	length	domain
R	1	A_1	N			
〃	2	A_2	N			
⋮	⋮	⋮	⋮			
〃	k	A_k	N			

H1から得る

表3.12　RRTBL

RRTBLは，QUELコマンド（define，retrieve）の結果リレーション（viewも含める）の情報を保持するためにある。

属　性　名	属性番号	ROLE	TYPE	LENGTH	説　　明
rrel-name	1	K	C	16	result relation名又はview name
degree	2	N	D	2	result relationのattribute数
type	3	N	C	2	V＝define　D＝delete R＝retrieve　PP＝replace A＝append
UP−REL	4	N	C	16	typeがVの場合，ビュースキーマ名を
head-adr	5	N	C	4	rrel-nameの結果がstoreされているSWFのhead address (byte address)
tail-adr	6	N	C	4	rrel-nameの結果がstoreされているSWFのtail-address (byte address)
qpt	7	N	P	2	この問合せのQ－treeへのpointer
qptV	8	N	P	2	query modificationされたQ－treeへのpointer

表3.13　RTBL

RTBL（range table）は，variable（組変数）とvelationとの対応表である。

属　性　名	属性番号	ROLE	TYPE	LENGTH	説　　明
var	1	K	C	2	variable名
rel-name	2	N	C	16	varに対応するrelation名
type	3	N	C	2	V：view　L：lcs-reletio R：snap-shot
work	4	N	D	1	work area

表3.14　RATBL

RATBLは，結果リレーション（viewも含む）の属性についての情報を保持している。

属　性　名	属性番号	ROLE	TYPE	LENGTH	説　　明
rrel-name	1	K	C	16	result relation名又はview名
att-no	2	K	D	2	属性番号
att-name	3	N	C	16	属性名
role	4	N	C	1	K：key N：non-key
type	5	N	C	1	C：文字 D：整数
length	6	N	D	4	バイト長
domain	7	N	C	16	domain名

b）Query　Modificationの実現

ビューを参照する問合せに対して，query modificationを行ない，LCSリレーションを参照する問合せに変形する。これを行う処理をQuery Modification処理と呼ぶ。Queny Modificationの処理概要は図3.48の様である。

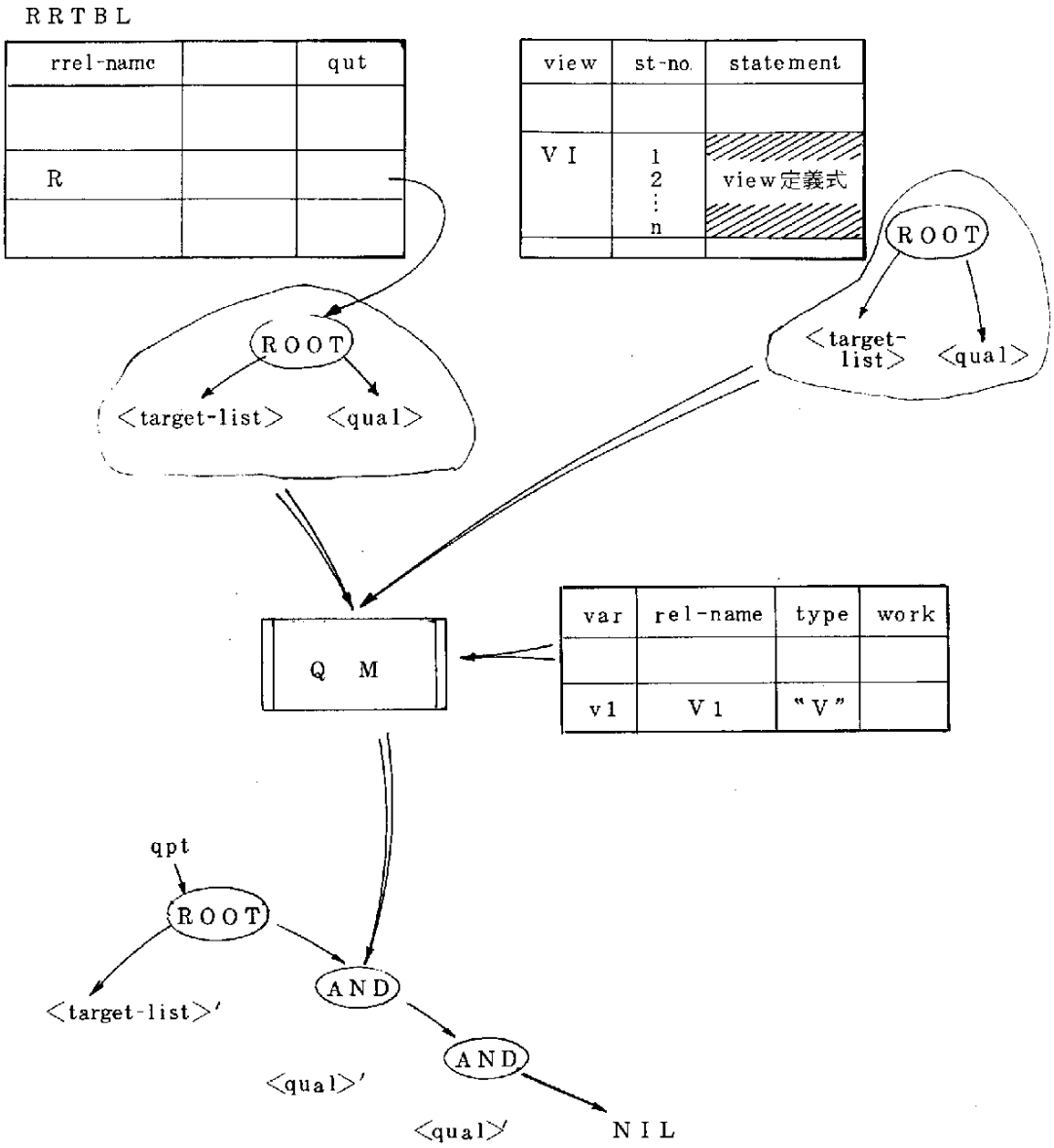

RRTBL
rrel-name
qut
R
view
st-no.
statement
VI
1
2
n
view定義式
ROOT
<target-list>
<qual>
ROOT
<target-list>
<qual>
QM
var
rel-name
type
work
v1
V1
"V"
qpt
ROOT
<target-list>'
AND
<qual>'
AND
<qual>
NIL

図3.4 8　QMの概要

ＱＭでは，まづ最初に関係参照木（ＲＲＴ）を生成する。生成されたＲＲＴを入力して，post-orderにたどり，参照しているビュー変数を求め，この求められたビュー定義式の木をコピーする。この時，済に同一ビュー変数がコピーされている時は，コピーを行なわない。

ビュー定義式の木をコピーする時に，ビュー定義木をin-orderにたどりながらビュー定義式内に参照している属性を見つける。この属性がＶＡＴＴＢＬ（表3.15，ビュー属性テブール）リレーションをサーチして見つかった場合は，この属性に対応するＶＡＴＴＢＬ内に格納されている木（属性節点又は式）へのポインタ（vatt-pt　属性の値）をこのコピーとして用いる。

即ち，ＶＡＴＴＢＬはビューの参照する属性とＬＣＳリレーション属性との対応表である。これにより，同一属性名に対して同一木を与えることができる。

ＤＡＴＴＢＬ

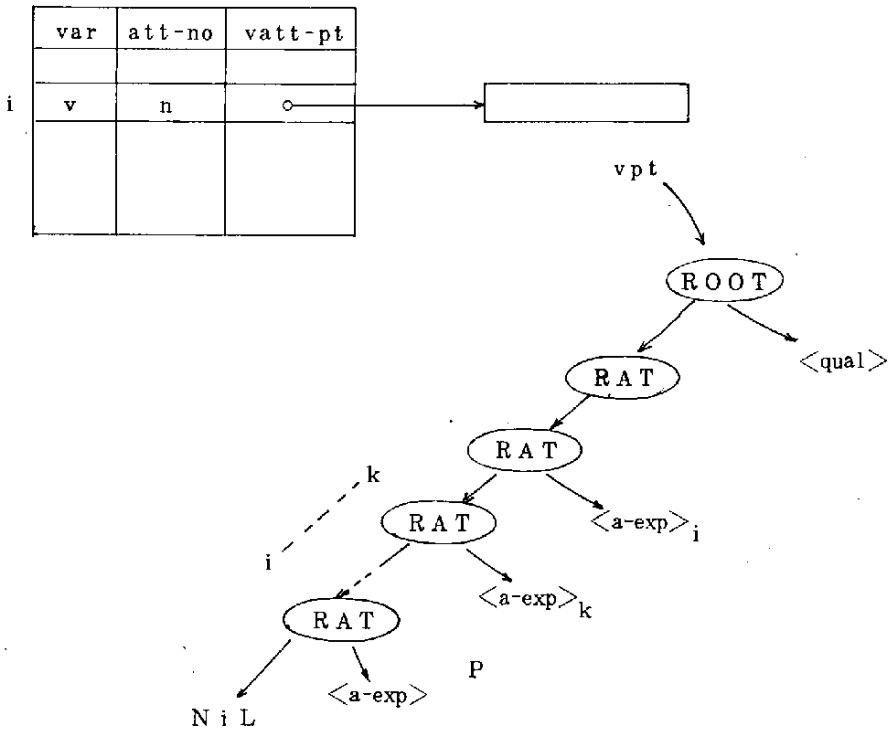


表3.15　ＶＡＴＴＢＬ

ＶＡＴＴＢＬはＱＭにおいて，あるＱ－treeが参照するview attributeの(view) variable，attribute番号と，この属性ノード（ＡＴＴＮ）へのポインタとの対応表である。

属　性　名	属性番号	ＲＯＬＥ	ＴＹＰＥ	LENGTH	説　　明
var	1	K	C	4	view variable名
att-no	2	K	D	2	attribute番号
vatt-pt	3	N	P		ＡＴＴＮへのポインタ

ＶＡＴＴＢＬを使用することにより，属性節点を一意に保つことができる。

また，ＲＲＴをpost-orderでたどりながら，ビュー定義木をコピーした木を次々に結合する。各時点で，query modificationされたビューのビュー属性とこれの定義式をＶＡＴＴＢＬにセットする。

最終的には，参照している全てのビュー定義木の条件式がビュー問合せの条件式に結合され，ビューで参照している属性は全てＬＣＳリレーションの属性に変換される。

b) ＬＣＳ問合せグラフ（ＬＱＧ）の実現

ＬＱＧは，Ｑ－treeと３つのテーブル（ＲＲＴＢＬ，ＲＡＴＢＬ，ＲＴＢＬ）を入力して生成される。この生成を行うプログラムをＬＱＧＰと呼ぶ。ＬＱＧＰの処理概要は図3.49の様である。

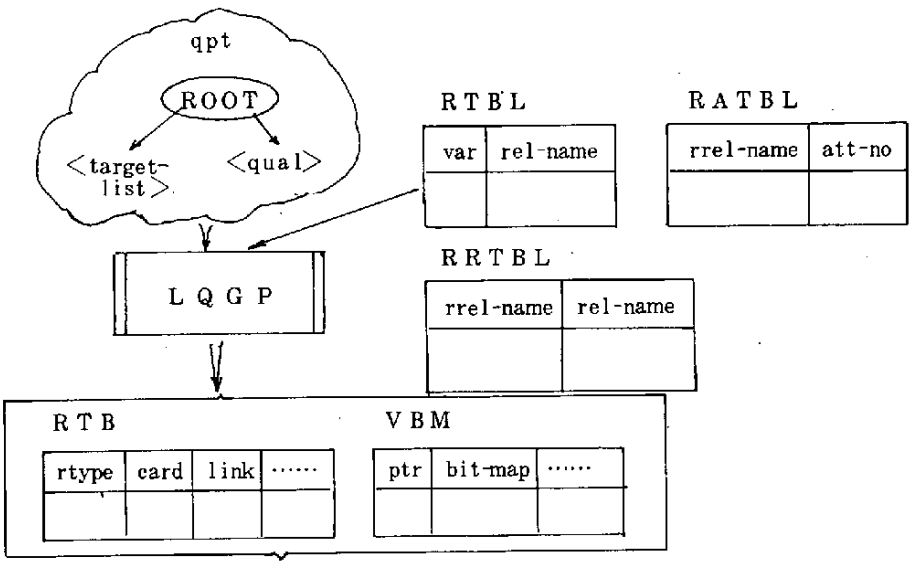


図3.49　ＬＱＧＰの処理概要

ＬＱＧは，ＱＵＥＬ問合せの変数を節点，目標属性，条件式を辺とするグラフである。

ＬＱＧは，変数及び制限式，結果属性に関する情報を格納したＲＴＢと，変数間の結合式についての情報を格納したＪＢＭとからなっている。ＪＢＭは，ビットマップによって，どの節点対についての結合式であるかを示す。

ＬＱＧＰは，次のことを行う。

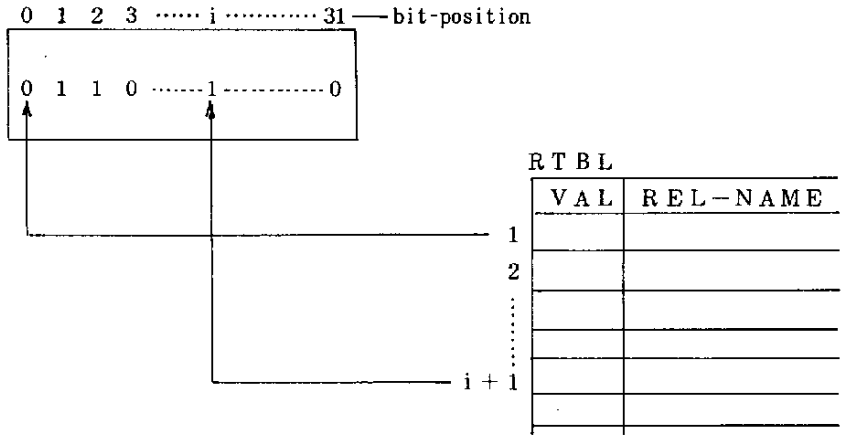
① 問合せの参照するリレーション（ＥＳＲ／ＲＳＲ）を明らかにし，ＲＴＢにセットする。このリレーションについての制限と結果属性のリストをＲＴＢにセットする。

② 節点の結合をＪＢＭと呼ばれるビットマップとして格納する。これはどのＬＣＳリレーション間に結合式があるかを示す。

まず最初に，目標属性及び条件式内の各conjunctionとこれを参照する組変数との対応表として使用している変数ビットマップ（Variable Bit Map，ＶＢＭ）について説明する。ＶＢＭのスキームは表3.16に示す。

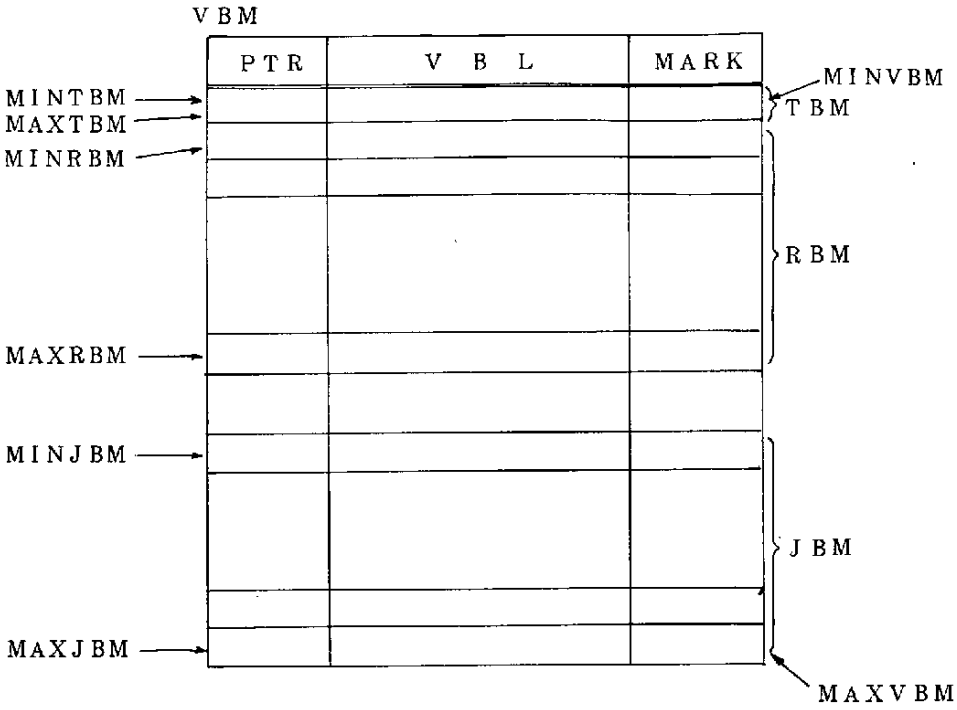
ＰＴＲは，目標属性又は<conjunction>の木へのポインタ値である。ここで，<conjunction>はclause又は，clauseのdisjunctionを意味する。

ＶＢＬは，３２bitから成るビットマップである。各ビット位置と組変数との対応は，ＲＴＢＬの組番号－１とこの組に格納された変数とによって表わされる。



このＶＢＭを入力して，各<conjunction>を，制限式と結合式に分類し，各々ＲＢＭ，ＪＢＭとしてまとめる。また目標属性リストからＴＢＭを作る。更に，ＲＢＭ，ＪＢＭ内で各々同一の変数を参照するもののconjunctionを作る。

ＴＢＭ，ＲＢＭ，ＪＢＭはＶＢＭのtable area内に作られる。



ＲＴＢは，Ｑ－tree，ＲＴＢＬ，ＲＢＭ及びＨＩ情報（ＥＳＲ，ＲＳＲ，ＡＴＴ）をもとにして，問合せ内で参照している変数に対応するＣＯＤＡＳＹＬの構文構造（レコード型セット型）をＲＴＢに格納する。

問合せ内で参照される全変数は，Ｑ－treeのＲＯＯＴ節点にチェインされているＢＶＬとＢＶＲとのＯＲをとることによって求められる。こうして求められたビットマップを用いて，ＲＴＢＬより問合せが参照するリレーション名を得ることができる。

次に，このリレーション名をＲＴＢに格納する。リレーションがレコード型かセット型か，又はリンクレコード型かはＨＩ情報をアクセスすることによって得られる。

更に，各リレーションに関する制限と結果属性情報，選択度等の情報をＲＢＭ，ＨＩを用いてセットする。こうして生成されたＲＴＢは，問合せ内に現われた全てのリレーションとこのリレーションに関連する情報とを格納していることになる。

表3.16　VBM(variable bit-map)

VBMは，問合せの<qualification>の各<conjunct>のbit-mapを格納するためのrelationである。

属性名	属性番号	ROLE	TYPE	LENGTH	説明
ptr	1	N	P		<conjunct>へのポインタ
type	2	N	D	1	<conjunct>の式のタイプ equi-join　EQJ(=1) その他　NN(=0)
VBL	3	N	C	4	<conjunct>のbit-map
mark	4	N	D	1	

表3.17　RTB

RTBは，variableに関する情報を格納するためのrelationである。

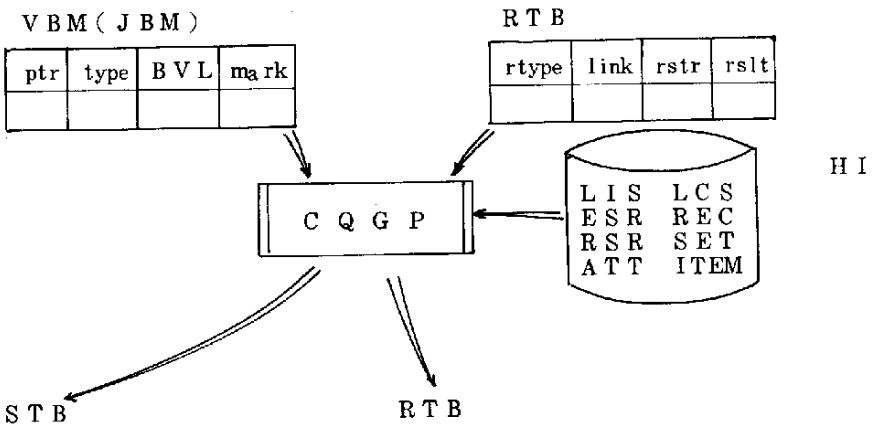
属性名	属性番号	ROLE	TYPE	LENGTH	説明
rtype	1	N	C	16	レコード型
card	2	N	D	4	cardinality
link	3	N	D	1	normal　(=0) link-type　(=1) set-type　(=2)
crstr	4	N	P		CALC(DNA)をもつequi-restriction treeへのポインタ(なければNIL)
rstr	5	N	P		CRSTRを除いたrestriction treeへのポインタ。(なければNIL)
cslct	6	N	F	4	CRSTRのselectivity(CRSTR=NILなり1)
rslct	7	N	F	4	RSTRのselectivity(RSTR=NILなり1)
rslt	8	N	P		result-attribute listへのポインタ
oca	9	N	D	4	レコード型内でアクセスされるオカーランス数(APGで使用)

属　性　名	属性番号	ＲＯＬＥ	ＴＹＰＥ	LENGTH	説　　明
nptr	１０	N	P		ＡＴノードへのポインタ （ＡＰＧで使用）
used	１１	N	D	1	ＯＮ　（＝１）使用中 ＯＦＦ（＝０）未使用
mark	１２	N	D	2	
type	１３	N	D	1	ＰＲＯ（＝０）　pro-node ＡＮＴＩ（＝１）ANTI-node

d) ＣＯＤＡＳＹＬ問合せグラフの実現

ＣＱＧは，節点としてレコード型の情報を，節点間の辺としてセット型の情報を表わしている。前者はＲＴＢリレーションに，後者はＳＴＢリレーションに保持される。

ＬＱＧからＣＱＧを生成するプロセスは，ＣＱＧＰと呼ばれ概要は次の図３.５０の様になっている。



stype	type	cree	mree	cunmac fivity	sort off	

rtype	card	link	crsts	rstr	cslct	rslct	rslt	oca	rptr	usel	mork

図３.５０　ＣＱＧＰの概要

ＣＱＧＰは，次のことを行なう。

イ）隠れ構造を明らかにする。

ロ）ＣＯＤＡＳＹＬ要素（即ち，ＳＴＢとＳＴＢ）にパフォーマンス情報をセットする。

ハ）unequi-join（不等価結合）のためにnon-set-typeを付加する。

ロ）リレーショナルモデル要素（リレーションと属性）をＣＯＣＡＳＹＬモデル要素（レコード型，セット型とデータ項目）へ変換する。

属性のデータ項目への変換について，次の点に注意する必要がある。

㋑ Ｒリレーションの属性は，本来これ自身のものでないものがある。このとき，他の節点（Ｅリレーション）にこれを移す。

㋺ 隠れ構造が存在するために，Ｒリレーション内の隠れ構造の節点は，隠れ構造が明らかになるまで，データ項目に変換できない。

このため，次のような変換方法をとる。

- Ｒリレーションの属性のデータ項目への変換は，隠れ構造が全て見つけられた時に行なう。
- ＲリレーションからＥリレーションへの属性の移動は，属性のままで行なう。
- ＲリレーションからＥリレーションへ移された属性は，移された時点で，データ項目へ変換する。

表3.18　ＳＴＢ（set-type table）

ＳＴＢには，ＣＱＧにおけるノード（レコード型）間のリンク情報が格納される。

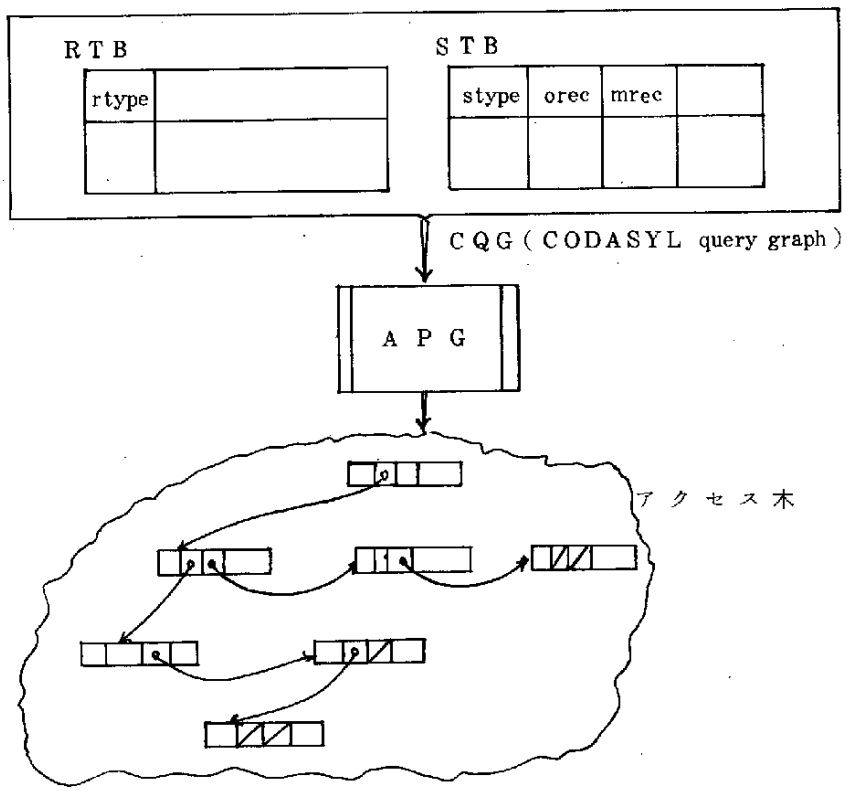
属　性　名	属性番号	ＲＯＬＥ	ＴＹＰＥ	LENGTH	説　　明
stype	1	K	C	16	set-type／non-set-type名
type	2	N	D	1	ONE（＝0）next-link TWO（＝1）owner link NSET（＝2）non-set-type
orec	3	N	D	1	owner recordのstoreされたＲＴＢのtuple番号
mrec	4	N	D	1	member recordのstoreされたＲＴＢのtuple番号
connectivity	5	N	D	4	orecとmrecの結合度
sort-att-no	6	N	D	2	sortされているmrecのsort keyのattribute番号
unique-att-no	7	N	D	2	uniqueなmrecのattribute番号

属　性　名	属性番号	ROLE	TYPE	LENGTH	説　　明
insertion	8	N	D	1	AUTOMATIC（＝0） MANUAL　　（＝1）
retension	9	N	D	1	MANDATORY（＝0） OPTIONAL　（＝1） FIXED　　（＝2）
mark	10	N	D	1	ON　（＝1） OFF（＝0）
ptr	11	N	P		join clause へのポインタ（non-set-type のときのみ…他はNIL）

e) アクセス木の実現

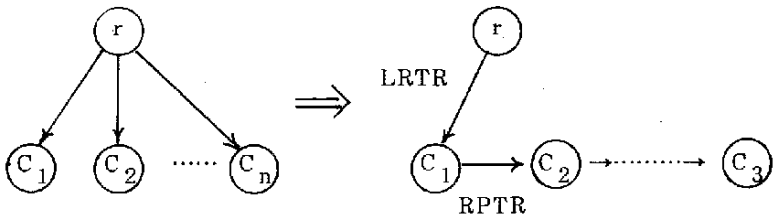
アクセス木は，セット辺によって連結な CQG_i の節点と辺から生成される。

CQG_i は，CQG内（RTBとSTB）でmarkがonの値をもつ節点と辺によって表わされる。



図'3.5.1　APGの概要

アクセス木（ＡＴ）は，自由節点空間（ＦＮＳ）内にＱ－treeと同様に生成される。
ＣＱＧから生成されるアクセス木は一般にn-ary treeであるが，これを２分木で表わす。



n-ary tree　　２分木

ＦＮＳ節点のＬＰＴＲは子節点へのポインタを表わし，ＲＰＴＲは兄弟節点のポインタを表わす。

ＡＴ節点の構成は次の通りである。

0	1	2	4	6	7	8	9	10
M	TYP	LPTR	RPTR	REC	SET	STYP	NTYP	NODE-NO

TYP　　ＡＴ節点を示す。（＝１０）
LPTR　　子節点の先頭節度へのポインタ
RPTR　　兄弟節点へのポインタ
REC　　ＲＴＢ番号
SET　　ＳＴＢ番号
STYP　　ＲＥＣはowner（=0）
　　ＲＥＣはmember（=1）
NTYP　　（=2）
NTYP　　NORM（=0）　normal節点
　　ＣＮ　（＝　）　合流節点
　　ＡＮＴＩ（=2）　反節点
　　ＰＲＯ　（=3）　pro節点
　　ＳＨ　（=4）　synonyn節点
NODE-NO　　NTYP=CN　のとき，最初に見つかったＣＮ節点番号
　　NTYP=ＰＲＯ，NTYP=ANTI　のとき　対のうち始めにtreeとして組み込まれた節点番号

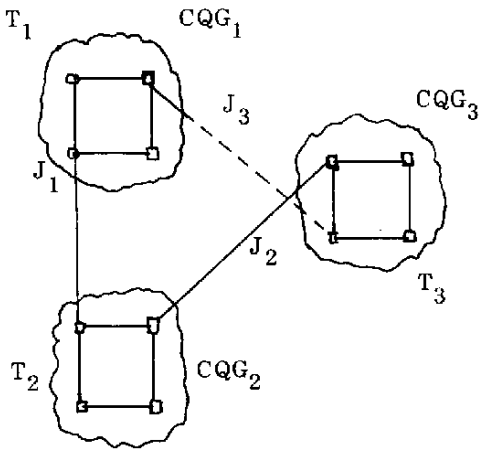
f）CQG_i 間の処理の実現

CQGの分割で分割されたCQG_i 間の処理を行う為にJQG（join query graph）を生成する。JQGは，分割された各CQG，間の結合関係を表わしたグラフである。JQGの節点12，各CQG_i を表わし，辺はCQG_i 間の結合式を表わす。JQGは，次の3つのリレーションから成る。

1） CQL

2） JQL

3） TATTBL



JQL

J_1		
J_2		
J_3		

TATTBL

T_1	
T_2	
T_3	

CQL

CQG_1	T_1	
CQG_2	T_2	
CQG_3	T_3	

図3.52　JQGの概要

CQLは，中間結果リレーション（T_1，T_2，……T_p）について次の情報を有している。

イ）中間結果リレーション名（T_i）

ロ）CQG#

ハ）CQG_i の参照する変数のbit map（BVM）

ニ）T_i からの目標属性リストへのポインタ

ホ）T_iが有している結合辺数

ＪＱＬは，T_iとT_j間の結合辺を表わしている。ＪＱＬは次の情報を有している。

イ）T_iとT_jの結合を表わすｂｉｔ-ｍａｐ（ＢＶＬ）

ロ）結合式へのポインタ

ＴＡＴＴＢＬには，次の情報が格納される。

イ）T_i

ロ）T_iの属性名

ハ）ロ）を生成する属性節点（目標属性節点）へのポインタ

ＣＱＬ，ＪＱＬ，ＴＡＴＴＢＬの構造は表3.19～3.21の様である。

表3.19　CQL

scheme　CQL　（CQG#, CBM, connect, rel−name, mark, rat−list）

CQLは、各CQGiのvariableのbit−map（CBM）, CQGiのjoin link数（connect）情報を保持する。

（in byte）

att-name	att-no	role	type	length	
CQG#	1	K	D	1	CQGiのi
CBM	2	N	C	4 (32bits)	bit−map
connect	3	N	D	1	ノートCQGiの持 join link数
rel-name	4	N	C		Ti
mark	5	N	D	1	
ratp	6	N	P	2	target−listへのpointer
schedule#	7	N	D	1	schedule#
jqpt	8	N	P	2	出力されるjoin fomulaへのpointer
PCQL#	9	N	D	1	schedule≠+1のCQLノート番号（次のCQGiへのpointer）

表3.20 JQL

scheme JQL （BVL, qpt, mark, rat−list）

JQLは、分割されたCQGi間の（BVL）のjoin formala cqpt）を表わしている。

（in byte）

att−name	att−no	role	type	length	
BVL	1	N	C	4 (32bits)	bit−map
qpt	2	N	P	2	join formulaへのpointer
mark	3	N	D	1	
ratp	4	N	P	2	target listへのpointer

表3.21 TATTBL

scheme TATTBL （rel−name, att−no, att−name, qpt）

（in byte）

att−name	att−no	role	type	length	
rel−name	1	K	C	16	中間結果リレーション名
att−no	2	K	D	1	attribute#
att−name	3	N	C	16	attribute name
qpt	4	N	P	2	target attributeへのpointer express

TATTBLは、中間結果リレーション（rel−name）と、その属性と、そのtarget expression （pointer）との対応表である。

g) DMLGの実現

DMLGは、アクセス木から、COBOL DMLプログラムを生成する。DML生成する。DML生成には、HI情報のDML I（DMLリレーション）を用いて行なう。

DMLGの処理概要は次の図3.5 3の様である。

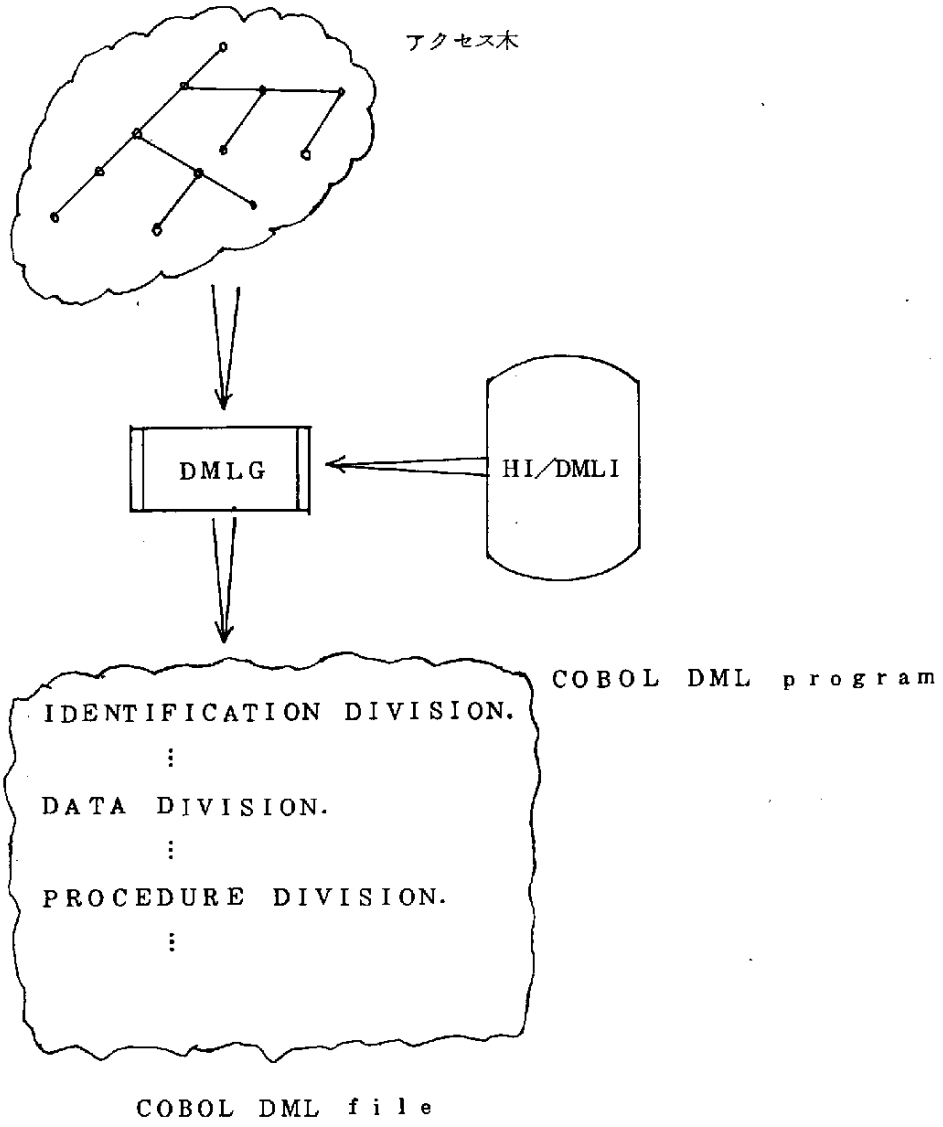


図3.5 3 DMLGの概要

DMLの生成手順を、図3.5 4に示す。

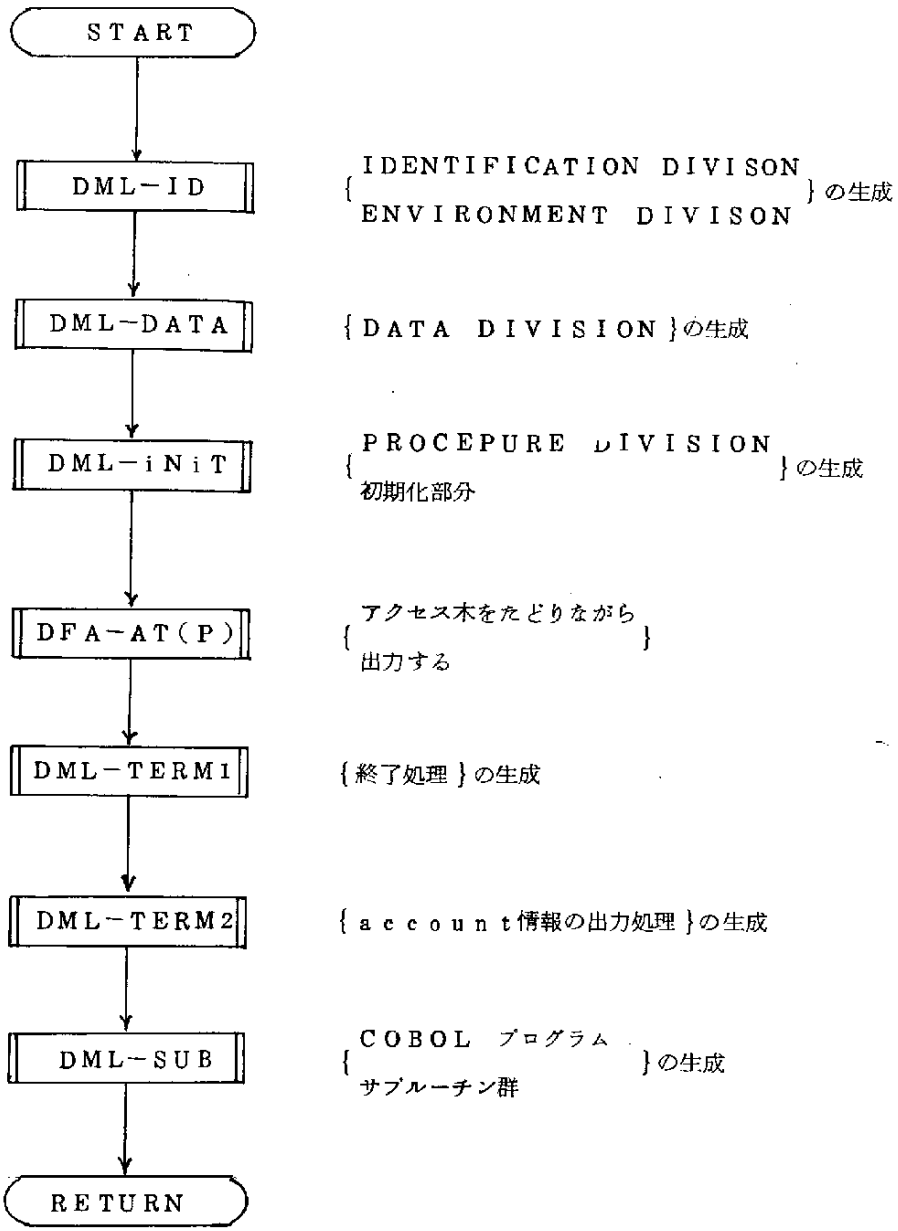


図3.5 4　DML生成手順

h）CODASYL　DBMSの起動方法の実現

CODASYL　DBMSの起動方法として、Jipnetの対話型プロトコル（ITP）を使用するITP　versionとメーカ提供のソフト、会話型プログラ

ミングファシリティ（ＩＰＦ）を使用するＩＰＦ　ｖｅｒｓｉｏｎを実現した。

(i) Ｍ－１７０Ｆ　ＩＴＰ　ｖｅｒｓｉｏｎ

ＩＴＰ　ｖｅｒｓｉｏｎでは、問合せ変換で生成されたＣＯＢＯＬ　ＤＭＬプログラムのコンパイルから実行までの一連のジョブを問合せ変換内で実行させ、検索結果をユーザに出力するために、ローカル通信機能（ＬＣＰ）を実現した。

ＬＣＰは、当協会で開発したＪｉｐｎｅｔのＩＴＰ（対話型プロトマル）を用いて、ＣＯＢＯＬ　ＤＭＬプログラムのコンパイルから実行までのジョブ制御文をリモートエントリシステム（ＲＥＳ）に送り実行させる。

ＩＴＰを通して、ＲＥＳにジョブを送信し、ジョブの結果を受信するための通信制御手順（プロトマル）は、次の図3.5 5の様である。

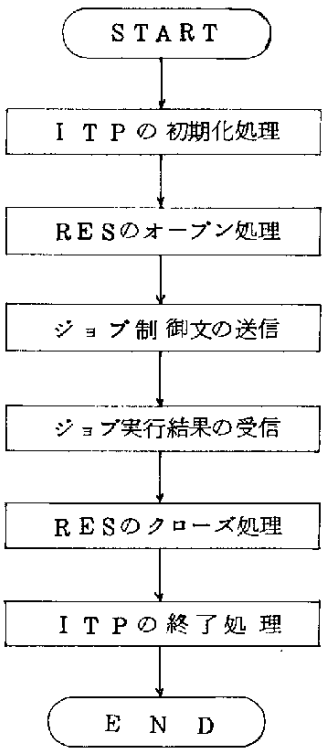


図3.5 5　ＩＴＰとの通信制御手順

ＩＴＰの初期化、送受信や終了処理を行うコマンドとして、次の５つのコマンドがある。

・ＩＴＯＰＮ　　ＩＴＰの初期設定を行う。

・ＩＴＩＮＴ　　ＩＴＰ割込み信号処理を行う。ＯＰＮ，ＣＬＳ，ＳＮＤ，ＲＣＶの

要求前の割込み信号を送る。

・ＩＴＳＮＤ　コマンド又はメッセージの送信処理を行う。コマンドはＣＬＳ，ＯＰＮであり、直前に割込み信号が出されている場合に限る。また、メッセージは最大値２５６バイド送信出来る。

・ＩＴＲＣＶ　メッセージの受信処理を行う。メッセージはメッセージ長とともに送られてくる。

・ＩＴＣＬＳ　ＩＴＰの終了処理を行う。

ローカル通信処理は、上の５つのコマンドを用いて実現されている。ローカル通信プロセッサの状態遷移図は、図３.５６の様である。

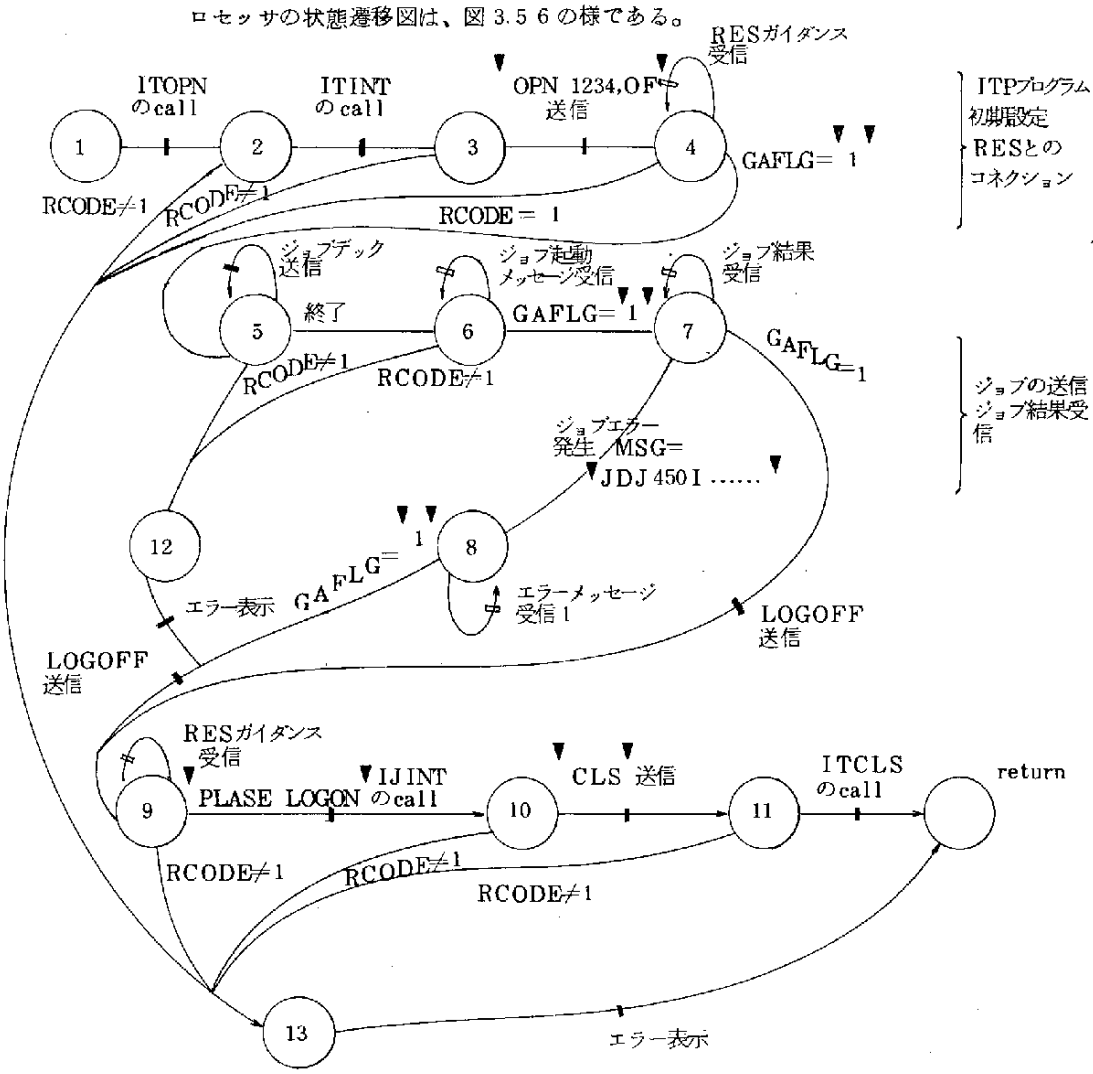


図３.５６　ＬＣＰの状態遷移図

(ii) M－１７０F　IPF　version

IPFを利用したCOBOL　DML　プログラムの実行を次の制限のもとに実行した。

・実行されるCOBOL　DML　プログラムは同時には１つしか実行できない。

・検索結果は端末にだけしか出力できない。

・検索結果をセーブする機能を持たない。

・IPE　version　を使用出来るTSSユーザは１ユーザのみである。

IPE　version　は次の図3.57の様に実現されている。

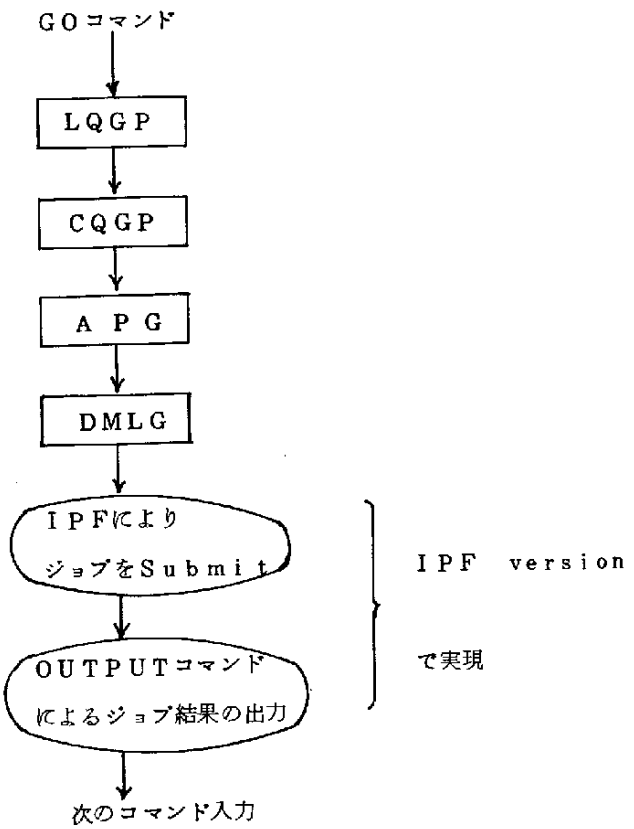


図3.57　M－１７０F　IPF　version

3.4.4　まとめと今後の課題

3.4節では、LDP－V 1.5を構成するスキーマ変換プロセッサ（HIP）と問合せ変換プロセッサ（QTP）の実現について述べた。

LDP－V 1.5は、まづ最初にスキーマ変換プロセッサ（HIP）とAIM上のデータベース

の開発から始まった。次に問合せ変換プロセッサ（ＱＴＰ）を開発した。

ＨＩＰ，ＱＴＰ及びデータベースの開発コスト及び使用環境は以下の様である。

1）ＨＩＰ（スキーマ変換プロセッサ）

ＨＩＰの開発コストは、次の様である。

・システム設計　　２人月

・コーディング・デバッグ　　２人月

・開発言語　　ＰＬ／Ｉ，ＣＯＢＯＬ

ステートメント数

ＰＬ／Ｉ　約3,５００　　ＣＯＢＯＬ　約５００

オブジェクトサイズ　約130ＫＢ

2）データベースの開発

データベースとしては、プロジェクト管理データベース〔3.5.1 評価用データベースを参照〕を開発した。

プロジェクト管理データベース（ＰＲＤＢＳ）は、まず最初はＡＩＭ上に実現し、後ちにＡＤＢＳ上に移植した。

ＰＲＤＢＳの開発コストは、次の様である。

・スキーマ設計　　１人月

・ローディングプログラム　　２人月

・データ作成　　１人月

・メンテナンス　　２人月

3）ＱＴＰ（問合せ変換プロセッサ）

ＱＴＰの開発コストは、次の様である。

・システム設計　　７人月

・コーディング・デバック　　１１人月

・開発言語　　ＰＬ／Ｉ

ステートメント数　　10,000

オブジェクトサイズ　約500ＫＢ

4）使用環境

ＨＩＰ、ＱＴＰは、ＰＬ／Ｉで書かれている。ＡＣＯＳ－７００では、ＰＬ／Ｉコンパイラの制限でＰＬ／Ｉ文数が約1,５００行までしかコンパイルできないという制限がある。この為、ＨＩＰとＱＴＰはＭ－１７０Ｆ（又はＭ－１６０）上で稼動している。Ｍ－１７０Ｆのオペレーティングシステムは ＦＡＣＯＭ　ＯＳ　ＩＶ／Ｆ４である。

サポートするＣＯＤＡＳＹＬデータベースシステムとしては、Ｍ－１７０Ｆ（Ｍ－１６０）

上のＡＩＭとＡＣＯＳ－７００上のＡＤＢＳである。ＡＤＢＳに対しては、ＡＣＯＳ用のＤＭＬ情報を使用して、Ｍ－１７０Ｆ上でＱＴＰを実行させ、ＣＯＢＯＬ　ＤＭＬ　プログラムを出力する。このＣＯＢＯＬ　ＤＭＬ　プログラムをＡＣＯＳの入力として検索を実行する。ＡＣＯＳ－７００　ＩＴＰ　ｖｅｒｓｉｏｎ　については、今後開発を行なう。

次に、Ｍ－１７０Ｆ　ＩＰＦ　ｖｅｒｓｉｏｎ　の使用例を示す。

```
READY
EXEC MMUI(LDP17GO)
 *** JDDBS LDP-V1.5 START...  15.37.45.460

#00010  RANGE ( E, EMPLOYEE)( PE, PROJ-EMP-LNK);
#00020  RANGE ( P, PROJECT);
#00030  RETRIEVE INTO T ( E.ENO, E.ENAME, E.EFNAME, E.BIRTH-PLACE)
#00040  ;
#00050  GO;

RETRIEVE INTO T ( ENO=E.ENO , ENAME=E.ENAME , EFNAME=E.EFNAME ,
     BIRTH-PLACE=E.BIRTH-PLACE ) ;

    THE ELAPSE TIME FOR RETRVP,QM..     51
    THE ELAPSE TIME FOR RQGP.....       15
    THE ELAPSE TIME FOR DQGP.....       17
    THE ELAPSE TIME FOR DFA1.....        2
    THE ELAPSE TIME FOR DML......      372
    THE ELAPSE TIME FOR SORT.....     1052
    THE ELAPSE TIME FOR DMLG.....     2452

---ESTIMATED ACCESS OCCURRENCE NUMBERS----- ---- OCAT =       69.000---
---ESTIMATED TOTAL SELECTIVITY             ----- ---- TSLCT =   0.10000E+01----
Q56250I JOB LDP0110Z SUBMITTED

    THE ELAPSE TIME FOR DQGD.....  9764
#00060  STATUS;
KE056211I JOB LDP0110Z EXECUTING
#00070  DUMP RTBL;

    ---- RTEL DUMP LIST ----
!VAR !REL_NAME          !TYPE!WORK!
-------------------------------------
!E   !EMPLOYEE          !L   !   0!
!FE  !PROJ-EMP-LNK      !L   !   0!
!P   !PROJECT           !L   !   0!
-------------------------------------

#00080  TSS;
)  ST
KE056211I JOB LDP0110Z EXECUTING
)  ST
KE056211I JOB LDP0110Z EXECUTING
```

ＬＣＳ問合せ

問合せ実行コマンド

各モジュールの経過時間

実行ジョブの状態表示コマンド

変数テーブル（ＲＴＢＬ）の表示コマンド

ＴＳＳモードへの切り換えコマンド

```
)  TEND;
#00090  STATUS;
KEQ55211I JOB LDP0110Z EXECUTING
#00100  STATUS;
KEQ55211I JOB LDP0110Z EXECUTING
#00110  STATUS;
KEQ55211I JOB LDP0110Z EXECUTING
#00120  STATUS;
KEQ55213I JOB LDP0110Z COMPLETED, WAITING FOR WRITER
#00130  OUTP;
 (|END ENAME  EFNAME  BIRTH-PLACE )
 TOKYO      MAKOTO                TAKIZAWA   00001
 IBARAGI    HIROYUKI              KOIKE      00002
 TOKYO      EIJI                  HAMANAKA   00003
 IBARAGI    SENICHI               NOMURA     00004
 YAMAGUCHI  KINKO                 YAMAMOTO   00005
 YAMAGATA   YOSHIHISA             OGAWA      00006
 TOKYO      SHOZO                 KAJI       00007
 GIFU       TETSUFUMI             ITO        00008
 OSAKA      YOSHIRO               AZUMA      00009
 HIROSHIMA  KAZUYOSHI             NISHIHARA  00010
 TOKYO      HIROMITSU             KAWASE     00011
 TOKYO      AKIHIRO               HOSHI      00012
 TOKYO      YOSHIAKI              NAGASAWA   00013
 TOKYO      MAKOTO                SUZUKI     00014
 TOKYO      TOMOAKI               MORO       00015
 IBARAGI    HIDEAKI               KONDOJI    00016
 KANAGAWA   YOSHIMITSU            HIRAI      00017
            HAJIME                USAMI      00018
 TOKYO      TOSHITAKE             KOBAYASHI  00019
 SAGA       AKIO                  SONODA     00020
 TOKYO      MINORU                YOKOTSUKA  00021
 MIYAGI     SHIGEMI               KOSEKI     00022
 TOKYO      NOTOKO                TSUKAMOTO  00023
 KYOTO      YUZURU                TANAKA     00024
 FUKUI      KOUICHI               TABATA     00025
 TOKYO      SETSUO                OHSUGA     00026
 IBARAGI    YUKA                  TOZAKI     00027
 TOKYO      RYOJI                 MIKI       00028
 FUKUI      YOSHIFUMI             MASUNAGA   00029
 HOKKAIDO   HIROSHI               KASENO     00030
 SAITAMA    YUKIO                 SATAKE     00031
            RYO                   YOSHIOKA   00032
            KIYOSHI               KANEDA     00033
            YOICHI                UENO       00034
            MITSUO                YOSHIDA    00035
 TOTTORI    MASATAKA              YOSHIZUMI  00036
            EIJI                  TAKEUCHI   00037
 TOKYO      ATSUSHI               MEGURO     00038
 CHINA      S. BING               YAO        00050
 USA        P. A.                 BERNSTEIN  00051
 USA        E.                    WONG       00052
 USA        M.                    STONEBRAKE 00053
 FRANCE     M.                    ADIBA      00054
 USA        M.                    HAMMER     00055
 USA        M.                    SIREU      00056
 USA        D.                    MCLEOD     00057
 USA        B.                    BERKOWITZ  00058
 USA        D.                    SHIPMAN    00059
```

検索結果の出力コマンド

検索結果

```
USA         B.                      NIAMIR     00060
USA         J. S.                   KUNIN      00061
USA         A.                      CHAN       00062
USA         K.                      KELLER     00063
USA         J. S.                   ROTHNIE    00064
USA         C.A.                    PAPADRMITR00065
USA         J.R.                    SWENSON    00066
USA         K.                      YOUSSEFI   00067
JAPAN       K.                      MIYASHITA  00068
JAPAN       S.                      KOYAMA     00069
JAPAN       E.                      MIYAMOTO   00070
JAPAN       T.                      TSUDA      00071
USA         L.A.                    ROWE       00072
USA         E.                      ALLMAN     00073
USA         P.                      KREPS      00074
USA         G.                      HELD       00075
GERMANY     E.                      HELHOLD    00076
USA         M.                      GOODMAN    00077
USA         S.A.                    SCHUSTER   00078
USA         D.                      DE JONG    00079
USA         A.                      HEVNER     00080
THE NUMBER OF LOCATED  OCCURRENCES  = 00000069
THE NUMBER OF ACCESSED OCCURRENCES  = 00000069
THE NUMBER OF OUTPUT   OCCURRENCES  = 00000069
THE ELAPSE TIME FOR INITIAL    = 0000000156
THE ELAPSE TIME FOR SRTRN      = 0000000001
THE ELAPSE TIME FOR FRTRN      = 0000000000
THE ELAPSE TIME FOR RESULT     = 0000000003
THE ELAPSE TIME FOR OUTPUT     = 0000000010
THE ELAPSE TIME FOR TERMINATE  = 0000000011
THE ELAPSE TIME FOR ACCESSING  = 0000000069
THE TOTAL TIME                 = 0000000250
```

COBOL DML プログラムの実行時のアカウント

```
#00140  RANGE ( T, T);
#00150  RETRIEVE INTO T2 ( TNAME= T.ENAME)
#00160  WHERE
#00170  T.ENO LE 20;
#00180  GO;
```

検索結果Tに対する問合せ

```
RETRIEVE INTO T2 ( TNAME=E.ENAME ) WHERE E.ENO  LE 20 ;

    THE ELAPSE TIME FOR RETRVP.OM..    48
    THE ELAPSE TIME FOR ROGP.......    17
    THE ELAPSE TIME FOR DOGP.......    21
    THE ELAPSE TIME FOR DFA1.......     2
    THE ELAPSE TIME FOR DML.....      451
    THE ELAPSE TIME FOR SORT.....    1062
    THE ELAPSE TIME FOR DMLG.......  3095

 ---ESTIMATED ACCESS OCCURRENCE NUMBERS----- ----  OCAT =       69.000--
 ---ESTIMATED TOTAL SELECTIVITY            ----- ---- TSLCT =   0.507E5E+00-
0562501 JOB LDP0110Z SUBMITTED

    THE ELAPSE TIME FOR DOGD.......  6230
#00190  STATUS;
***
```

```
STATUS;
KE0562111I JOB LDP0110Z EXECUTING
#00200  DUMP RTBL;

     ---- RTBL DUMP LIST ----
!VAR.!REL_NAME          !TYPE!WORK!
-------------------------------------
!E   !EMPLOYEE          !L   !   0!
!PE  !PROJ-EMP-LNK      !L   !   0!
!P   !PROJECT           !L   !   0!
!T   !T                 !R   !   0!
-------------------------------------

#00210  RANGE ( R, REPORTR)(ER, EMP-REP-LNK)(RJ, REP-SUBJ)(J, SUBJECT);
#00220  DEFINE DDBS-REP ( R.@R, R.RND  ) WHERE
#00230  R.@R=RJ.@R AND RJ.@J = J.@J AND
#00240  J.SUBJECTN ="DISTRIBUTED DATABASE SYSTEMS"
#00250  ;
#00260  RANGE ( DR, DDBS-REP);
#00270  STATUS;
KE0562131 JOB LDP0110Z COMPLETED, WAITING FOR WRITER
#00280  OUTP;
 ( ENAME )

TAKIZAWA
KOIKE
HAMANAKA
NOMURA
YAMAMOTO
OGAWA
KAJI
ITO
AZUMA
NISHIHARA
KAWASE
HOSHI
NAGASAWA
SUZUKI
MORO
KONGOJI
HIRAI
USAMI
KOBAYASHI
SONODA
THE NUMBER OF LOCATED  OCCURRENCES  = 00000069
THE NUMBER OF ACCESSED OCCURRENCES  = 00000069
THE NUMBER OF OUTPUT   OCCURRENCES  = 00000020
THE ELAPSE TIME FOR INITIAL    = 0000000095
THE ELAPSE TIME FOR SRTRN      = 0000000000
THE ELAPSE TIME FOR FRTRN      = 0000000000
THE ELAPSE TIME FOR RESULT     = 0000000001
THE ELAPSE TIME FOR OUTPUT     = 0000000003
THE ELAPSE TIME FOR TERMINATE  = 0000000019
THE ELAPSE TIME FOR ACCESSING  = 0000000074
THE TOTAL TIME                 = 0000000192
```

ビュー定義

```
#00290  RETRIEVE INTO
#00300  T3 ( E.END, DR.RNO)
#00310  WHERE
#00320  E.@E= ER.@E AND ER.@R = DR.@R AND
#00330  ER.AUTH-NO = 1
#00340  ;
#00350  GO;
```

ビュー問合せ

問合せ変形後の問合せ

```
RETRIEVE INTO T3 ( END=E.END , RNO=R.RNO ) WHERE E.@E  = ER.@E
     AND ER.@R  = R.@R  AND ER.AUTH-NO  = 1  AND R.@R  = RJ.@R
     AND RJ.@J  = J.@J  AND J.SUBJECTN  = "DISTRIBUTED DATABASE
   SYSTEMS" ;

  THE ELAPSE TIME FOR RETRVP,GM..  2384
  THE ELAPSE TIME FOR RQGP.......  1839
  THE ELAPSE TIME FOR DQGP.......    47
  THE ELAPSE TIME FOR DEAL.......     5
  THE ELAPSE TIME FOR DML........  1581
  THE ELAPSE TIME FOR SORT.......  1042
  THE ELAPSE TIME FOR DMLG.......  3625
---ESTIMATED ACCESS OCCURRENCE NUMBERS----- ----  OCAT =       42.475---
---ESTIMATED TOTAL SELECTIVITY             ----- ---- TSLCT =  0.14706E-02----   KE
056250I JOB LDP0110Z SUBMITTED

    THE ELAPSE TIME FOR DQGO.......  6779
#00360  DISPLAY;
```

入力した問合せの表示

```
!LINE!     STATEMENT
-------------------------------------------------------------------------------
!   30!RETRIEVE INTO T ( E.END, E.ENAME, E.EFNAME, E.BIRTH-PLACE)
!   40!;
!  150!RETRIEVE INTO T2 ( TNAME= T.ENAME)
!  160!WHERE
!  170!T.END LE 20;
!  220!DEFINE DOSS-REP ( R.@R, R.RNO  ) WHERE
!  230!R.@R=RJ.@R AND RJ.@J = J.@J AND
!  240!J.SUBJECTN ="DISTRIBUTED DATABASE SYSTEMS"
!  250!;
!  290!RETRIEVE INTO
!  300!T3 ( E.END, DR.RNO)
!  310!WHERE
```

```
| 320|E.@E= ER.@E AND ER.@R = DR.@R AND
| 330|ER.AUTH-NO = 1
| 340|;
```

```
#00370  EDIT 300 T4 ( E.ENO, DR.RNO, ER.AUTH-NO);
#00380  DISPLAY 290,340;
```

——— 入力文の編集

```
|LINE|     STATEMENT
```

```
| 290|RETRIEVE INTO
| 300|T4 ( E.ENO, DR.RNO, ER.AUTH-NO)
| 310|WHERE
| 320|E.@E= ER.@E AND ER.@R = DR.@R AND
| 330|ER.AUTH-NO = 1
| 340|;
```

```
#00390  EDIT 330 ( ER.AUTH-NO = 1 OR
#00400  EDIT 331   ER.AUTH-NO = 2 );
#00410  RENUM;
#00420  DISPLAY 290,400;
```

} 入力文の編集

```
|LINE|     STATEMENT
```

```
#00430  DISPLAY;
```

```
|LINE|     STATEMENT
```

```
|  10|RETRIEVE INTO T ( E.ENO, E.ENAME, E.EFNAME, E.BIRTH-PLACE)
|  20|;
|  30|RETRIEVE INTO T2 ( TNAME= T.ENAME)
|  40|WHERE
|  50|T.ENO LE 20;
|  60|DEFINE DDBS-REP ( R.@R, R.RNO  ) WHERE
|  70|R.@R=RJ.@R AND RJ.@J = J.@J AND
```

```
!  801J.SUBJECTN ="DISTRIBUTED DATABASE SYSTEMS"
!  901;
! 1001RETRIEVE INTO
! 1101T4 ( E.END, DR.RNO, ER.AUTH-NO)
! 1201WHERE
! 1301E.BE= ER.BE AND ER.BR = DR.BR AND
! 1401( ER.AUTH-NO = 1 OR
! 1501   ER.AUTH-NO = 2 )
! 1601;
```

```
#00440  STATUS:
KEQ5&213I JOB LDP0110Z COMPLETED, WAITING FOR WRITER
#00450  OUTP;
 ( ( RNO ( ENO ) ) )
 HAMAE78A00003
 HAMAE78B00003
 HAMAE79 00003
 HAMAE80 00003
 JDDBS78 00003
 JDDBS79 00003
 JDDBS80A00001
 TAKIM78 00001
 TAKIM79A00001
 TAKIM79B00001
 TAKIM79C00001
 TAKIM79D00001
 TAKIM79E00001
 TAKIM80A00001
 TAKIM80B00001
 TAKIM80C00001
 TAKIM80D00001
 TAKIM80E00001
 TAKIM80F00001
 BERNP77A00051
 BERNP77B00051
 BERNP78A00051
 BERNP78B00051
 BERNP78C00051
 BERNP79A00051
 BERNP80A00051
 BERNP80B00051
 GOODN79A00077
 HAMMM79B00055
 HEVNA78A00080
 HEVNA78B00080
 HEVNA79 00080
 STONM77A00053
 STONM75A00053
 STONM76A00053
```

← 結果の出力

```
STONM77A00053
STONM78A00053
STONM78B00053
STONM79A00053
STONM79B00053
STONM80A00053
WONGE77A00052
YAOS 80A00050
THE NUMBER OF LOCATED  OCCURRENCES  = 00000181
THE NUMBER OF ACCESSED OCCURRENCES  = 00000176
THE NUMBER OF OUTPUT   OCCURRENCES  = 00000086
THE ELAPSE TIME FOR INITIAL    = 0000000127
THE ELAPSE TIME FOR SRTRN      = 0000000004
THE ELAPSE TIME FOR FRTRN      = 0000000000
THE ELAPSE TIME FOR RESULT     = 0000000001
THE ELAPSE TIME FOR OUTPUT     = 0000000009
THE ELAPSE TIME FOR TERMINATE  = 0000000019
THE ELAPSE TIME FOR ACCESSING  = 0000000268
THE TOTAL TIME                 = 0000000428

#00460  GO;      ← 2回目の実行

RETRIEVE INTO T4 ( ENO=E.ENO , RNO=R.RNO , AUTH-NO=ER.AUTH-NO )
     WHERE E.@E  = ER.@E  AND ER.@R  = R.@R  AND (ER.AUTH-NO  = 1
      OR ER.AUTH-NO  = 2 ) AND R.@R  = RJ.@R  AND RJ.@J  = J.@J
     AND J.SUBJECTN  = "DISTRIBUTED DATABASE SYSTEMS" ;

    THE ELAPSE TIME FOR RETRVP,OM..   147
    THE ELAPSE TIME FOR RQGP.......  1857
    THE ELAPSE TIME FOR DQGP.......    27
    THE ELAPSE TIME FOR DFA1.......     6
    THE ELAPSE TIME FOR DML......    1697
    THE ELAPSE TIME FOR SORT.....    1073
    THE ELAPSE TIME FOR DMLG.......  3777

---ESTIMATED ACCESS OCCURRENCE NUMBERS----- ----  OCAT =       69.753---
---ESTIMATED TOTAL SELECTIVITY             ----- ---- TSLCT =   0.29412E-02---
Q562501I JOB LDP0110Z SUBMITTED
    THE ELAPSE TIME FOR DQGD.......  6454
#00470  DEFINE DDBS-PROJ ( P.@P, P.PNO, P.PNAME)
#00480  WHERE
#00490  P.@P=PJ.@P AND PJ.@J = J.@J AND
#00500  J.SUBJECTN ="DISTRIBUTED DATABASE SYSTEMS";
VARIABLE=PJ     WAS NOT FOUND IN RTBL
ERROR IN SCAN, ERR_ID IS  3- 8
SYNTAX ERROR IN QUAL
#00510  DUMP RTBL;

    ---- RTBL DUMP LIST ----
IVAR IREL_NAME        ITYPEIWORK!
-----------------------------------
IE   IEMPLOYEE        IL   I   0I
IPE  IPROJ-EMP-LNK    IL   I   0I
IP   IPROJECT         IL   I   0I
```

```
 IT   IT                IR   |   01
 IR   IREPORTR          IL   |   01
 IER  IEMP-REP-LNK      IL   |   01
 IRJ  IREP-SUBJ         IL   |   01
 IJ   ISUBJECT          IL   |   01
 IDR  IDDBS-REP         IR   |   01
```

```
#00520  RANGE ( PJ, PROJ-SUBJ);
#00530  DEFINE DDBS-PROJ ( P.@P, P.PNO, P.PNAME)
#00540  WHERE
#00550  P.@P=PJ.@P  AND  PJ.@J = J.@J AND
#00560  J.SUBJECTN ="DISTRIBUTED DATABASE SYSTEMS";
#00570  RANGE ( DP, DDBS-PROJ);
#00580  DEFINE ERP1( E.@E, R.@R, P.@P)
#00590  WHERE
#00600  E.@E=ER.@E AND ER.@R=R.@R AND E.@E=PE.@E AND
#00610  PE.@P = P.@P AND ER.AUTH-NO =1
#00620  ;
#00630  RANGE ( A, ERP1);
#00640  RETRIEVE INTO T5( DR.RNO, DP.PNO )
#00650  WHERE
#00660  DR.@R = A.@R AND DP.@P=A.@P
#00670  ;
#00680  GO;
```

```
RETRIEVE INTO T5 ( RNO=R.RNO , PNO=P.PNO ) WHERE R.@R  = R.@R  AND

      P.@P  = P.@P  AND E.@E  = ER.@E  AND ER.@R  = R.@R  AND E.@E

       = PE.@E  AND PE.@P  = P.@P  AND ER.AUTH-NO  = 1  AND P.@P

     = PJ.@P  AND PJ.@J  = J.@J  AND J.SUBJECTN  = "DISTRIBUTED

     DATABASE SYSTEMS"  AND R.@R  = RJ.@R  AND RJ.@J  = J.@J  AND

     J.SUBJECTN  = "DISTRIBUTED DATABASE SYSTEMS" ;   ← 修正された問合せ

    THE ELAPSE TIME FOR RETRVP.QM..  2055
    THE ELAPSE TIME FOR RQGP.......  3524
    THE ELAPSE TIME FOR DQGP.......    47
    THE ELAPSE TIME FOR QFA1.......    11
    THE ELAPSE TIME FOR DML......    2690
    THE ELAPSE TIME FOR SORT.....    1270
    THE ELAPSE TIME FOR DMLG.......  5118

 ---ESTIMATED ACCESS OCCURRENCE NUMBERS----- ----  OCAT =        0.000---
 ---ESTIMATED TOTAL SELECTIVITY             ----- ---- TSLCT =   0.00000E+00----
Q56250I JOB LDP0110Z SUBMITTED

    THE ELAPSE TIME FOR DQGD.......  8326
```

```
#00690  STATUS;
KEQS6211I JOB LDP0110Z EXECUTING
#00700  STATUS;
KEQS6211I JOB LDP0110Z EXECUTING
#00710  DUMP RTBL;

     ---- RTBL DUMP LIST ----
|VAR |REL_NAME          |TYPE|WORK|
-------------------------------------
|E   |EMPLOYEE          |L   |  0|
|PE  |PROJ-EMP-LNK      |L   |  0|
|P   |PROJECT           |L   |  0|
|T   |T                 |R   |  0|
|R   |REPORTR           |L   |  0|
|ER  |EMP-REP-LNK       |L   |  0|
|RJ  |REP-SUBJ          |L   |  0|
|J   |SUBJECT           |L   |  0|
|DR  |DDBS-REP          |R   |  0|
|PJ  |PROJ-SUBJ         |L   |  0|
|DP  |DDBS-PROJ         |R   |  0|
|A   |ERP1              |R   |  0|
-------------------------------------

#00720  DUMP RRTBL;

--------  RRTBL DUMP  --------
|RREL-NAME         |DEG|TYPE|  OPT|OPTV|
-----------------------------------------
|T                 |  4|R   |   16|   0|
|T2                |  2|R   |   97|   0|
|DDBS-REP          |  2|R   | 1061|3951|
|T3                |  2|R   | 1461|   0|
|T4                |  2|R   | 3171|   0|
|DDBS-PROJ         |  3|R   | 3151|4051|
|ERP1              |  2|R   | 3391|4191|
|T5                |  2|R   | 3751|4261|
-----------------------------------------

#00730  DUMP RATBL;

|RREL-NAME         |ATT#|ATT-NAME          |ROLE|TYPE|LNGTH|DOMAIN
|T                 |   1|ENO               |  0I|  DI|    5|
|T                 |   2|ENAME             |  0I|  CI|   10|
|T                 |   3|EFNAME            |  0I|  CI|   20|
|T                 |   4|BIRTH-PLACE       |  0I|  CI|   10|
|T                 |   1|ENO               |  0I|  CI|    5|
|T                 |   2|ENAME             |  0I|  CI|   10|
|T                 |   3|EFNAME            |  0I|  CI|   20|
|T                 |   4|BIRTH-PLACE       |  0I|  CI|   10|
|T2                |   1|TNAME             |  0I|  CI|   10|
|T2                |   1|TNAME             |  0I|  CI|   10|
|DDBS-REP          |   1|@R                |  1I|  CI|    8|
|DDBS-REP          |   2|RNO               |  0I|  CI|    8|
|T3                |   1|ENO               |  0I|  DI|    5|
|T3                |   2|RNO               |  0I|  CI|    8|
|T3                |   1|ENO               |  0I|  DI|    5|
|T3                |   2|RNO               |  0I|  CI|    8|
|T4                |   1|ENO               |  0I|  DI|    5|
|T4                |   2|RNO               |  0I|  CI|    8|
```

結果リレーション
テーブル表示

結果属性テーブル表示

```
|T4                |   3|AUTH-NO           |   0|   D|     2|
|DDBS-PROJ         |   1|@P                |   1|   C|     8|
|DDBS-PROJ         |   2|PNO               |   0|   C|     8|
|DDBS-PROJ         |   3|PNAME             |   0|   C|   100|
|DDBS-PROJ         |   1|@P                |   1|   C|     8|
|DDBS-PROJ         |   2|PNO               |   0|   C|     8|
|DDBS-PROJ         |   3|PNAME             |   0|   C|   100|
|ERP1              |   1|@E                |   1|   C|     8|
|ERP1              |   2|@R                |   1|   C|     8|
|ERP1              |   3|@P                |   1|   C|     8|
|T5                |   1|RNO               |   0|   C|     8|
|T5                |   2|PNO               |   0|   C|     8|
|T5                |   1|RNO               |   0|   C|     8|
|T5                |   2|PNO               |   0|   C|     8|
#00740  STATUS;
KE0562131 JC3 L0P0110Z COMPLETED, WAITING FOR WRITER
#00750  OUTP;
 ( ( RNO ( AUTH-NO END ) ) )
 HAMAE78A 100003
          200001
 HAMAE78B 100003
 HAMAE79  100003
          200001
 HAMAE80  100003
          200001
 JDDBS78  100003
          200004
 JDDBS79  100003
          200004
 JDDBS80A 100001
          200014
 TAKIM78  100001
          200003
 TAKIM79A 100001
          200003
 TAKIM79B 100001
          200003
 TAKIM79C 100001
 TAKIM79D 100001
 TAKIM79E 100001
 TAKIM80A 100001
          200003
 TAKIM80B 100001
          200003
 TAKIM80C 100001
 TAKIM80D 100001
 TAKIM80E 100001
 TAKIM80F 100001
          200003
 BERNP77A 100051
          200077
 BERNP77B 100051
          200059
 BERNP78A 100051
          200064
 BERNP78B 100051
```

```
                200059
     BERNP78C 100051
                200059
     BERNP79A 100051
                200077
     BERNP80A 100051
                200059
     BERNP80B 100051
                200059
     GDDDN779A 100077
                200051
     PAMMM79B 100055
                200059
     HEVNA78A 100080
                200050
     HEVNA78B 100080
                200050
     HEVNA79  100080
     STONM73A 100053
     STONM75A 100053
     STONM76A 100053
                200073
     STONM77A 100053
                200076
     STONM78A 100053
     STONM78B 100053
     STONM79A 100053
     STONM79B 100053
     STONM80A 100053
     WONGE77A 100052
     YAOS 80A 100050
     THE NUMBER OF LOCATED  OCCURRENCES  = 00000252
     THE NUMBER OF ACCESSED OCCURRENCES  = 00000204
     THE NUMBER OF OUTPUT   OCCURRENCES  = 00000181
     THE ELAPSE TIME FOR INITIAL   = 0000000132
     THE ELAPSE TIME FOR SRTRN     = 0000000000
     THE ELAPSE TIME FOR FRTRN     = 0000000000
     THE ELAPSE TIME FOR RESULT    = 0000000005
     THE ELAPSE TIME FOR OUTPUT    = 0000000015
     THE ELAPSE TIME FOR TERMINATE = 0000000012
     THE ELAPSE TIME FOR ACCESSING = 0000000283
     THE TOTAL TIME                = 0000000447

#00760  END;                                    ←―― 終了コマンド

*** JDDBS LDP-V1.5 END ***  15.57.35.289
```

次にＬＤＰ－Ｖ1.5を実現する上で問題となった点をまとめると次の様になる。

1) プログラムの巨大化

問合せ変換のプログラムが巨大化（ＰＬ／Ｉ文数で約10,000行）するにつれて、コンパイルを行うのに応答時間が２０分程度もかかるようになってしまった。この為、メンテナンスの効率が悪くなった。また、変数名やサブルーチン名との管理が大変になった。

メンテナンスの効率化の為に、途中から外部サブルーチン化を行なった。これにより、メンテナンスの効率化が計られた。

2) ＤＭＬパターンのデバッグ

生成されＣＯＢＯＬ　ＤＭＬ　プログラムのデバッグに苦労した。ＤＭＬパターン内にデバッグツールの組み込みが無かったので、デバッグ作業に労力がかかった。

ＤＭＬパターン内にデバッグ作業を容易にする為のＣＯＢＯＬ文の組み込みが必要である。

3) ＱＵＥＬ問合せの入力修正機能

ＱＵＥＬ問合せ入力では、行単位に入力させているので、入力ミスに対する行単位の修正機能では使いにくかった。この為、結合数の多い問合せは、ほとんどバッチ入力に頼らざるを得なかった。

問合せ入力に対して、ＴＳＳ端末に提供されるＦｕｌｌ　ｓｃｒｅｅｎモードの修正機能の使用を検討する。

4) ＣＯＤＡＳＹＬ　ＤＢＭＳの起動方法

ＣＯＤＡＳＹＬ　ＤＢＭＳを起動する方法として、まづ最初にローカル通信処理（Ｆ－160　ＩＴＰ　ｖｅｒｓｉｏｎ）を実現した。しかし、ローカル通信処理はオーバヘッドが大きかった〔3.5.2　性能評価参照〕。また、生成されたＣＯＢＯＬ　ＤＭＬプログラムのコンパイルから実行結果が返ってくる間（平均約４～５分）、端末が占有され、ユーザはその間何も出来ないという問題点がある。この為、ＣＯＤＡＳＹＬ　ＤＢＭＳの起動方法を再検討することが必要である。

ＣＯＤＡＳＹＬ　ＤＢＭＳを起動する方法としては、次の３つを考える。

1) ＩＴＰを利用したローカル通信処理

2) ＩＰＦを利用した方法

3) 各ＤＭＬのＳＹＳＴＥＭ　ｃａｌｌを用いたｉｎｔｅｒｐｒｅｔｅｎ方法

ＩＴＰを利用したローカル通信処理は、Ｊｉｐｎｅｔを使用することにより、ＡＣＯＳとＦＡＣＯＭのリソースを共有することが出来るという利点がある。分散型データベースシステムにおける同一ノード内の問合せ処理とみることができる。しかし、例えば、ＦＡＣＯＭ内だけで問合せ処理を行うことを考えると、ネットワークの起動や負荷が大きいという欠点をもつ。

また、ＩＰＦはＭ－１６０からＭ－１７０Ｆにリプレースされたことにより利用可能となったソフトである。ＩＰＦを使用することで、ＣＯＢＯＬ　ＤＭＬプログラムの実行とＡＩＭ上のデータベースの検索結果を問合せ変換内で出力することが出来る。但し、これはＩＰＦを使用でき

るＦＡＣＯＭユーザに限る。このＩＰＦの利用に関しては、実用的なシステムとして、ＣＯＤＡＳＹＬデータベースシステムに対するリレーショナル検索インタフェースシステムＲＩＳＣＯＳ〔付記１〕を検討している。

３）は、ＣＯＢＯＬ　ＤＭＬプログラムのコンパイル方法でなく、各ＤＭＬのＳＹＳＴＥＭ ｃａｌｌを用いたｉｎｔｅｒｐｒｅｔｅｒ方法である。この問題点としては、各ＤＭＬのＳＹＳＴＥＭ　ｃａｌｌがユーザに公開されていない点である。

3.5　評　　価

ＣＯＤＡＳＹＬデータベースに対するリレーショナルインタフェース（ＬＤＰ－Ｖ1.5）を性能面と利用面から評価を行う。

評価目的としては、性能評価、利用面からの評価について各々、次の様な点にある。

(1) 性能評価（ｐｅｒｆｏｒｍａｎｃｅ評価）

ＬＤＰ－Ｖ1.5の性能として、応答時間の評価を行なう。ＬＤＰ－Ｖ1.5を構成する各モヂュール（ＲＥＴＲＶＰ、ＬＱＧＰ、ＣＱＧＰ、ＡＰＧ、ＤＭＬＧ）の経過時間を測定し、ＬＤＰ－Ｖ1.5の応答時間に対してｂｏｔｔｌｅ　ｎｅｃｋとなっているモヂュールを明確にする。

ｂｏｔｔｌｅ　ｎｅｃｋとなっているモヂュールについては改良を検討し、その効果の評価を行なう。

また、ＬＤＰ－Ｖ1.5でインプリメントしたアクセスパス生成の最適化に対して、最適化の仮定の検証、最適化の為のオーバヘッド、最適化の効果や選択度の評価を行なう。

その他に、格納構造を変えた場合のアクセス効率の評価とＣＯＤＡＳＹＬ型のＤＢＭＳ間（ＡＩＭ、ＡＤＢＳ）の互換性を確認する。

3.5.1　評価用データベース

評価用データベースとしては、ＰＲＤＢＳ（プロジェクト管理データベースシステム）を用いる。ＰＲＤＢＳは、当協会のプロジェクトを管理するデータベースである。従業員とプロジェクト、プロジェクトと研究テーマ、スポンサーとの関係等を表わしている。ＰＲＤＢＳは、当協会のＭ－１７０Ｆ上のＡＩＭとＡＣＯＳ－７００上のＡＤＢＳを用いて、各々インプリメントされている。

Ａ　スキーマ

ＰＲＤＢＳの論理構造は、図3.58の様になっている。

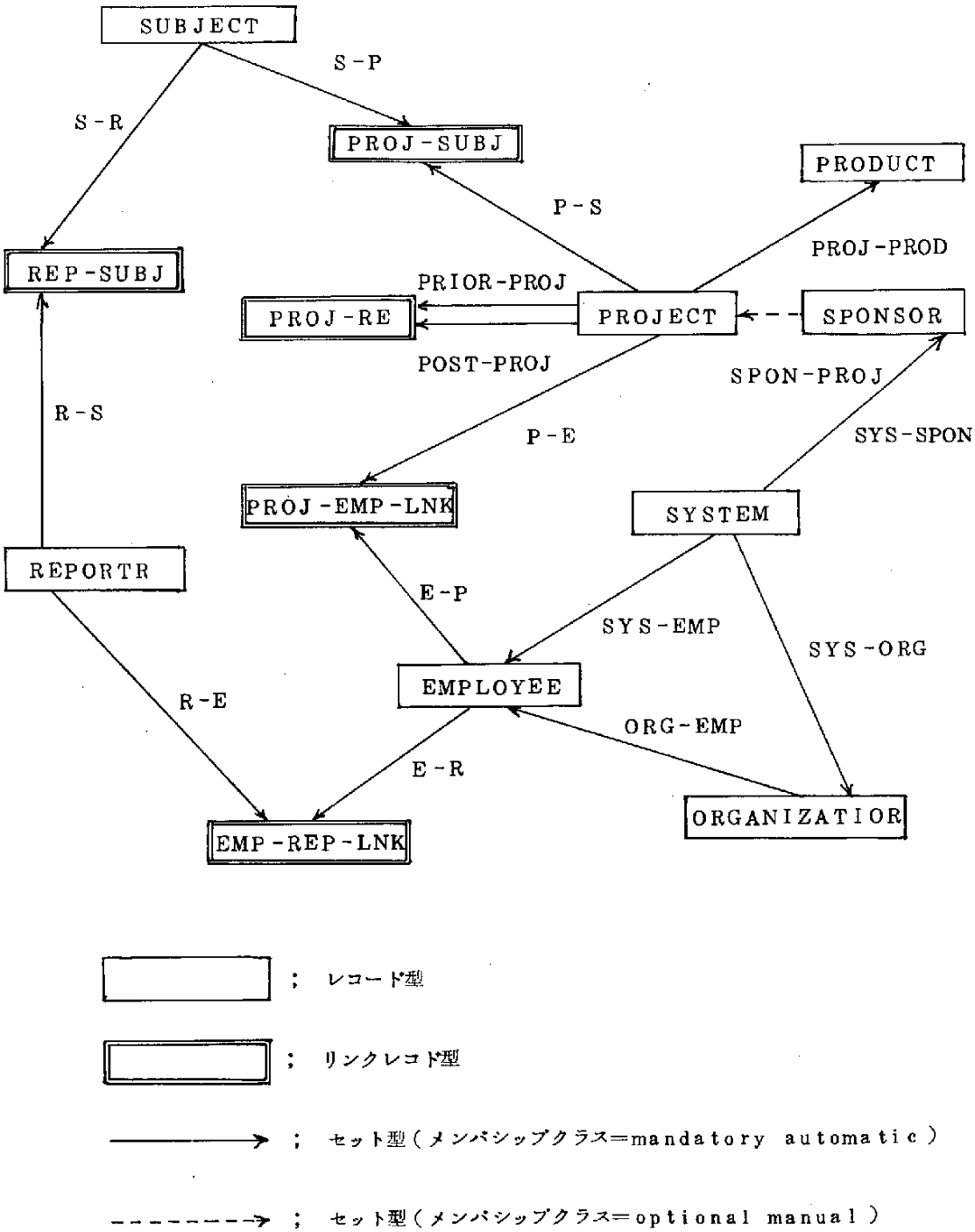


図3.5.8　論理スキーマ構造図

ＰＲＤＢＳの物理構造は、図3.5９の様になっている。

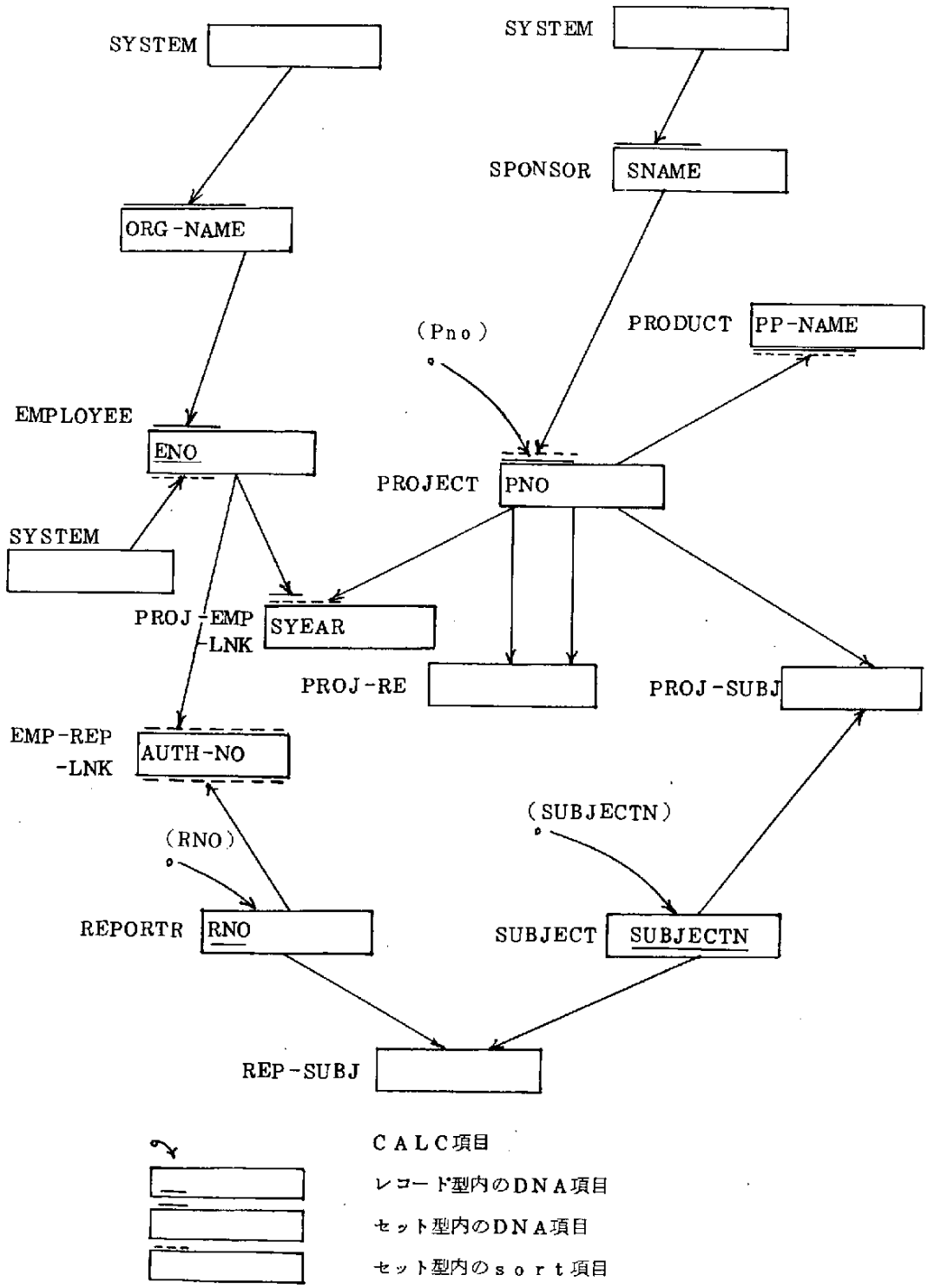


図 3.5 9　物理構造図

B　データベースの容量

ＰＲＤＢＳは、１２のレコード型と１６のセット型から成る。各レコード型のオカーランス数は、表3.２２の様になっている。

表3.２２　レコードオカーランス数

レコード型名	リンクレコード型（Ｙｅｓ／Ｎｏ）	オカーランス数
EMPLOYEE	Ｎｏ	６８
ORGANIZATIOR	Ｎｏ	１５
PRODUCT	Ｎｏ	９
SUBJECT	Ｎｏ	１３６
PROJECT	Ｎｏ	１４
REPORTR	Ｎｏ	８４
SPONSOR	Ｎｏ	４
PROJ－EMP－LNK	Ｙｅｓ	６２
EMP－REP－LNK	Ｙｅｓ	１５５
REP－SUBJ	Ｙｅｓ	６２５
PROJ－SUBJ	Ｙｅｓ	２０１
PROJ－RE	Ｙｅｓ	５

各セット型のオカーランス数は、表３.２３の様になっている。

表3.２３　セットオカーランス数

セット型名	セットオカーランス数
SYS－ORG	１
SYS－EMP	１
SYS－SPON	１
ORG－EMP	１５
SPON－PROJ	４
PROJ－PROD	９
E－P	６２
E－R	１５５
R－E	１５５
R－S	６２５
S－R	６２５
S－P	２０１
P－S	２０１
P－E	６２
PRIOR－PROJ	５
POST－PROJ	５

C　AIM（FACOM）とADBS（ACOS）の相違

a）コード体系の相違

FACOMではEBCDICコード、ACOSではJISコードを各々使用している。コード体系により、文字の大小関係がFACOMとACOSで異なる。従って、同じデータをソートすると、ソート後のデータの順番がFACOMとACOSで異なってしまう。

また、FACOMとACOSの間で同じ媒体（MT）を利用するためには、コード変換が必要になる。例えば、FACOMで出力したCOBOL DMLプログラムは、ACOSでコード変換しないとコンパイラーに入力することができない。

b）コンパイル方式の相違

データベースを利用するCOBOL DMLプログラムに対して、FACOMでは、プリコンパイラーによってDMLをCOBOLサブルーチンコールに変換することが必要である。

ACOS は、COBOL74コンパイラーに直接入力してコンパイルすることが出来る。

c）COBOLの相違

(1) 順編成ファイル

順編成ファイルに対する入出力命令に対するエラー条件の書き方が異なる。ACOSでは、各ファイルに対してファイルの状態コードを示すファイルステータスをデータ定義部で定義する必要がある。エラー条件は、このステータスのコードで判定する。

ADBS

WRITE　レコード名
IF　ファイルーステータス＝10
無条件命令

AIM

WRITE　レコード名　INVALID KEY　無条件命令

(2) SECTION化

COBOLプログラムの区分化で、FACOMとACOSでは次の様に異なる。

ACOSでは、区分化を行なう場合、常駐部分に対してもSECTIONの指定を行なわなければならない。FACOMでは、常駐部分にSECTIONを指定する必要はない。

d）データベースにおけるデータ構造の相違

(1) メンバシップクラス

AIMでは、メンバシップクラスがサポートされていないのに対し、ADBSではMANDATORY AUTOMATICとOPTIONAL MANUALがメンバシップクラスとしてサポートされている。

この為、データベースのオカーランスのローディング時にADBSではローディングの条件としてメンバシップクラスを考慮しなければならない。

(2) 単項ｓｅｔ

ＡＩＭでは、ＳＹＳＴＥＭを親とする単項ｓｅｔがサポートされているが、ＡＤＢＳではサポートされていない。ＡＤＢＳでは、ユーザ自身がＳＹＳＴＥＭに対応するレコード型をつくり、単項ｓｅｔとして管理する必要がある。

(3) ＣＡＬＣ項目

ＡＩＭでは、ＣＡＬＣ項目に対して値の重複が許される（ＤＡ　ＣＡＬＣ）が、ＡＤＢＳではＣＡＬＣ項目には値の重複が許されない（ＤＮＡ　ＣＡＬＣ）。

(4) セット型の構造

ＡＩＭでは、セット型としてリング形態とリスト形態がある。ＡＤＢＳでは、セットタイプとしてチェイン表現とアレイ表現がある。リング形態とチェイン形態、リスト形態とアレイ形態が各々対応する。

ＡＩＭのリング形態の場合、親リンクとＰＲＩＯＲリンクはオプションであるが、ＡＤＢＳのチェイン形態では必須となる。

(i) リスト形態（アレイ表現）

リスト形態とは、一つの親レコードが、関係のある一つ又は複数の子レコードの全ポインタを持つことにより関係づけられている形態である。

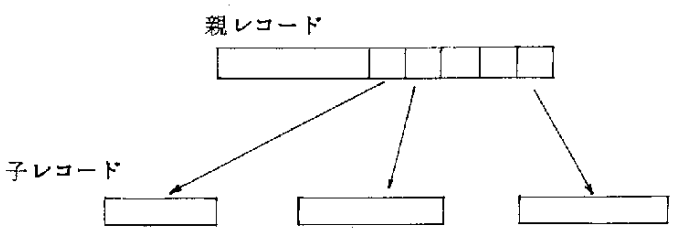


図3.60　リスト形態（アレイ表現）

(ii) リング形態（チェイン表現）

リング形態とは、一つの親レコードと、一つ又は複数の子レコードがそれぞれの関係のあるレコードの格納アドレスをレコード制御部により関係づけられている形態である。

親レコードは、関係のある子レコード群の内で、一つの子レコードを指すポインタをもつ。そして、親レコードにより指された子レコードは、その親レコードと関係のある他の子レコードを指すポインタ（ＮＥＸＴポインタ）をもつ。親レコードや前の子レコードへのポインタ（ＯＷＮＥＲポインタやＰＲＩＯＲポインタ）はＡＩＭではオプションである。ＡＤＢＳにおけるチェイン表現では、ＮＥＸＴ、ＯＷＮＥＲとＰＲＩＯＲポインタは必須である。

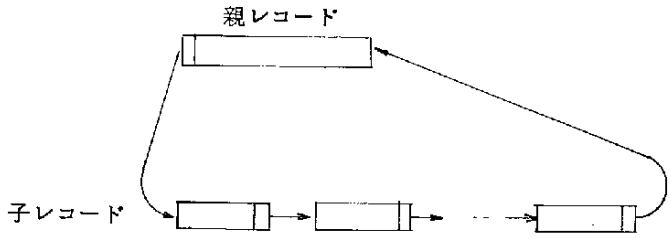


図 3.6 1　リング形態（チェイン表現）

(5)　ＳＥＴ　ＳＥＬＥＣＴＩＯＮ

ＡＩＭでは、ＳＥＴ　ＳＥＬＥＣＴＩＯＮはセット型の形態の選択を表す。即ち、ＳＥＴ　ＳＥＬＥＣＴＩＯＮを指定すると、セット型としてリスト形態になる。即ち、ＣＯＤＡＳＹＬに準拠していない。

ＡＤＢＳでは、ＤＢＭＳがセットオカーランスを自動的に選択する為の指定を行う。これは、ＣＯＤＡＳＹＬに準拠している。

(6)　エリアの相違

ＡＩＭでは、データベースの格納領域の分割単位として、論理的にはレンジ、物理的にはデータセットがある。レンジは、サブレンジに分割することが出きる。サブレンジへの格納は指定されたキー値によって実行される。

ＡＤＢＳでは、データベースの格納領域の分割単位としてエリア、物理的にはファイルがある。

ＡＩＭとＡＤＢＳでは、レンジとエリア、データセットとファイルが各々対応している。

ＡＩＭでは複数のレンジを１つのデータセットに割り当てられる。しかし、ＡＤＢＳでは、エリアとファイルは１対１に割り当てなければならない。

(7)　スキーマの機密保護の相違

ＡＤＢＳでは、スキーマやサブスキーマに対して変更、翻訳、ロックの変更や出力機能に対してロックをかけることができる（ＣＯＤＡＳＹＬに準拠している）。

ＡＩＭでは、これに対応する機密保護機能は無い。

(8)　データ定義言語の相違

データ定義言語（ＤＤＬ）の相違としては次の様な点がある。

・データ定義と物理装置定義の分離

ＡＤＢＳは、データ定義（ＤＤ）と物理装置定義（ＤＭＣ）が分離されている。

ＡＩＭでは、データ定義と物理装置定義をスキーマとして１つの中に定義しなければならない。

ADBS

```
SCHEMA NAME IS スキーマ名          ┐
        ⋮                          ├ データ定義
END-SCHEMA.                        ┘

SCHEMA NAME IS スキーマ名          ┐
        ⋮                          ├ 物理装置定義
END-DMCL.                          ┘
```

AIM

```
SCHEMA NAME IS スキーマ名 FOR DB;  ┐
        ⋮                          ├ データ定義
DATASET NAME IS データセット名;    ┐ │
        ⋮                          ├ 物理装置定義
END.                               ┘ ┘
```

また、AIMではスキーマ内にデータセット名及びそのデータセットが存在するボリューム名を指定しなければならない。

ADBSでは、物理定義では使用するファイルのコードを指定するだけで、実際のファイルへの割当ては実行時に制御文によって行なわれる。

e) DML言語の相違

(1) サブスキーマの宣言

AIMではENVIRONMENT DIVISIONで、ADBSではDATA DIVISIONで各々サブスキーマを宣言する。即ち、ADBSはCODASYLに準拠している。

AIM

```
ENVIRONMENT     DIVISION.
CONFIGURATION     SECTION.
SUBSCHEMA-NAME.     "PROJSUB1".
```

ADBS

```
DATA                DIVISION.
SUS-SCHEMA          SECTION.
DB  PROJSUBI        WITHIN   PROJDBS.
```

(2) READY命令

ADBSでは、READY命令に対してUSAGE-MODE IS { RETRIEVAL / UPDATE }

のどちらかを指定する。

ＡＩＭでは、ＵＳＡＧＥ－ＭＯＤＥはＣＯＢＯＬ　ＤＭＬプログラムでは指定できない。

ＡＩＭ

```
READY　〔レンジ名〕
```

ＡＤＢＳ

```
READY　〔エリア名〕

USAGE-MODE IS { RETRIEVAL }
              { UPDATE    }
```

ＡＩＭによるデータベースのアクセスモードは、スキーマなどと一緒にシステムにあらかじめ登録して置く必要がある。

(3) データベースキー

データベースキーを格納する項目に対して、ＡＩＭでは文字タイプ８桁のデータ項目であれば良い。

しかし、ＡＤＢＳではデータベースキーを格納する項目に対して、ＵＳＡＧＥ　ＩＳ　ＤＢ－ＫＥＹを指定しなければならない。

ＡＩＭ

```
04　DBKEY　PIC　X(08).
```

ＡＤＢＳ

```
04　DBKEY　USAGE　IS　DB-KEY.
```

(4) 現在子（Ｃｕｒｒｅｎｃｙ）

ＡＩＭでは、実行単位（ｒｎｎ－ｕｎｉｔ）の現在子だけを保持している。セットの現在子を保持しない。この為、ＡＩＭでは、セットオカーランスの終了を検出後の次のＦＩＮＤ　ＮＥＸＴ命令で、セット型の親レコードオカーランスが位置付けられる。

ＡＤＢＳでは、実行単位、セット型、レコード型、エリアに対する現在子を保持している。セットオカーランスの終了を検出後の次のＦＩＮＤ　ＮＥＸＴ命令でもセットオカーランスの終了が検出される。

(5) ＡＣＣＥＰＴ命令

ＡＤＢＳでは、カレントレコードのデータベースキーを取り出すＡＣＣＥＰＴ命令があるが、ＡＩＭにはない。

ＡＩＭでは、連絡領域内のデータ項目ＰＧＣＳの値を取り出すことに対応する。

ＡＩＭ

```
MOVE　PGCS　OF　FCOM　TO　LO230.
```

ＡＤＢＳ

```
ACCEPT　LO230　FROM　CURRENCY.
```

(6) ＦＩＮＤ命令

(i) データベースキーによるＦＩＮＤ命令

データベースキーによるＦＩＮＤ命令は、ＡＩＭにはサポートされていない。ＡＩＭでは、連絡領域内のデータ項目ＰＧＣＳに値をセットし、ＦＩＮＤ　ＣＵＲＲＥＮＴ命令によって実行される。

ＡＩＭ

```
MOVE  LO230  TO  PGCS  OF  FCOM.
FIND  CURRENT.
```

ＡＤＢＳ

```
FIND  レコード名；DB−KEY  IS  LO230.
```

(ii) セット内のＦＩＮＤ命令

ＡＩＭでは、セット型がリスト形態の場合だけ、ＦＩＮＤ　{ＬＡＳＴ / ＦＩＲＳＴ / 整数}　レコード名　ＷＩＴＨＩＮセット名。によるＦＩＮＤがサポートされる。

ＡＤＢＳでは、セット型がチェイン表現やアレイ表現のどちらでもサポートされる。

(iii) ＣＡＬＣキーに対するＦＩＮＤ　ＤＵＰＬＩＣＡＴＥ命令

ＡＤＢＳでは、ＣＡＬＣキーに対して重複が許されていない為に、ＣＡＬＣキーに対するＦＩＮＤ　ＤＵＰＬＩＣＡＴＥ命令はサポートされていない。

(7) ＧＥＴ命令

ＡＩＭのＧＥＴ命令は、位置決め（ＦＩＮＤ）とレコードのワークエリアへの転送を同時に行なう。

ＡＤＢＳでは、ＧＥＴ命令は、位置決めをＦＩＮＤ命令で実行した後、データベースエリアへのレコードの転送を行なう。

(8) ＳＥＴ命令

ＡＩＭではデータベースキーの転送は、ＭＯＶＥ命令で実行する。ＡＤＢＳではデータベースキーの転送はＳＥＴ命令で実行しなければならない。データベースキー１をデータベースキー２に転送する命令は次の様になる。

ＡＩＭ

```
MOVE  データベースキー1  TO  データベースキー2
```

ＡＤＢＳ

```
SET   データベースキー2  TO  データベースキー1
```

(9) 例外コードの相違

ＡＩＭとＡＤＢＳにおける検索ＤＭＬ文（ＦＩＮＤ文）に対する例外条件は表3.24～3.25の様である。

表3.24　AIMの例外条件（FIND文）

例外コード	データベース例外条件
11	セット又はレンジの終りになった。
12	メンバレコードがない。又はオーナレコードがない。
13	指定されたプライムキー値をもつエントリレコードがない。

表3.25　ADBSの例外条件（FIND文）

FIND	状態コード	データベース例外条件
05	02100	レルムまたはセットの最後を検出した。
05	02400	レコード選択表現を満すレコードが見つからない。
05	03100	レルム、セットタイプまたはレコードタイプのカレントが空（NULL）である。

AIMとADBSの例外条件の例として、セット又はレンジ（あるいはレルム）の終わりになったという条件に対しては、各々次の様に書く。

AIM

```
FIND FIRST レコード名 WITHIN レンジ名 RANGE.
IF DB-EXCEPTION IS 11　無条件命令.
```

ADBS

```
FIND FIRST レコード名 WITHIN レルム名.
IF DB-STATUS ="05021 00"　無条件命令.
```

f）データベース全体的な相違

AIM、ADBSともにネットワーク構造をサポートするデータベースシステムであるが、AIMはADBSに比べるとCODASYL 73に準拠していない。

AIMではメンバシップクラスがOPTIONAL MANUALだけが許されたデータ構造になっている。この為セット関係は全てユーザの責任で管理しなければならない。DML言語もOPTIONAL MANUALに対するものだけが提供される（e.g.セット関係は自動storeがない）。又AIMではアクセスを効率的に行なわせる為に、CODASYL仕様には無い、INDEXやサブレンジなどを提供している。

ADBSは、CODASYL 73に準拠しているが、SYSTEMを親とする単項セットをサポートしていない。又、データベースに対するファイルの管理は通常のファイルと同じ管理システムによって管理される。この為、データベースの保護や共有は通常のファイルの

共有や保護と同一に扱われる。

D 問 題 点

a) ＡＩＭ、ＡＤＢＳでは、ネットワーク構造を持った複雑なスキーマに対してのデータベースのオカーランスローディングのサポートがない。

b) ＤＢＭＳの相違によるデータベースローディングの問題点

ＡＩＭでは、セット型にメンバシップクラスがＯＰＴＩＯＮＡＬ　ＭＡＮＵＡＬしかない。このため、レコードオカーランスと関係性のローディングを別々に実行できる。即ち、レコードオカーランスをstoreしてからconnectできる。

しかし、ＡＤＢＳでは、セット型にメンバシップクラスが存在する。このため、ＭＡＮＤＡＴＯＲＹ　ＡＵＴＯＭＡＴＩＣのメンバシップクラスがある場合は、レコードオカーランスと関係性のローディングを別々に実行できない。メンバシップクラスがＭＡ（ＭＡＮＤＡＴＯＲＹ　ＡＵＴＯＭＡＴＩＣの略）の子レコードオカーランスはセット型の親レコードオカーランスがローディングされた後でないとローディングが出来ない。

即ち、スキーマ構造によって、レコードオカーランスのローディングに対してあるスケジュールが必要となる。

ＡＤＢＳに対するデータベースのローディングに対して、次の方法でスケジュールを決定した。

ローディング方法

i) ＭＡの子レコードになっていないレコードに対して、オカーランスのローディングを行う。

ii) ＭＡの子レコードで、ｉ）によって親レコードがローディングされたものについてローディングを行なう。この時、関係性のローディングも同時に行なう。

iii) ii）で親レコードがローディングされたものについて順次子レコードをローディングしていく。

iv) 全てのレコードがローディングされた後、ＯＭ（ＯＰＴＩＯＮＡＬ　ＭＡＮＵＡＬの略）の関係性をローディング（ｃｏｎｎｅｃｔ）する。

また、ＡＤＢＳでは、単項ｓｅｔがサポートされていない。そこでＳＹＳＴＥＭに対してレコード型を作り、１レコードオカーランスのレコード型として管理する。

c) 関係性のメンテナンスに多大な労力を要した。

関係性を各レコード型のオカーランスに一連番号を割り当て、その一連番号によってレコードオカーランス間の関係性を表わした。関係性が一連番号同志なので関係の正しさが陽にはわかりにくかった。

この為、関係性をメンテナンスするために検索プログラムを実行させ、関係性のチェックを行った。検索結果から間違った関係性を表わすデータを発見するのが容易ではなかった。

3.5.2 性 能 評 価

A 目　的

性能評価の目的は、次の４つである。

a　性能評価

b　最適化の評価

c　格納構造とアクセス効率の評価

d　ＣＯＤＡＳＹＬ型ＤＢＭＳ間の互換性の達成

a）性能評価

ＬＤＰ－Ｖ１.５の性能評価として、問合せ変換機能について評価を行なう。

問合せ変換機能を

・ＱＵＥＬ問合せ入力からＣＯＢＯＬ　ＤＭＬプログラムを生成する機能

・ＣＯＢＯＬ　ＤＭＬプログラムを実行する機能（ローカル通信機能）

とに分けて評価を行なう。

①　Ｐｅｒｆｏｒｍａｎｃｅボトルネックの明確化とその改良及び効果

問合せ入力からＣＯＢＯＬ　ＤＭＬプログラムを生成する機能について評価を行なう。

ここでＰｅｒｆｏｒｍａｎｃｅとしては、各モデュールに対する経過時間を測定する。ボトルネックとしては、モデュールの経過時間が全体に対して占める割合の大きいものと考える。

まづ初めに、ボトルネックとなっているモデュールを明確にする。次にボトルネックとなっているモデュールに対してモデュールの処理を検討し、改良を行なう。次いで、この改良に対してモデュールの経過時間の減少を評価する。

②　ローカル通信処理の性能評価

ＣＯＢＯＬ　ＤＭＬプログラムを実行するために、ＩＴＰを用いた通信に要する経過時間（ＣＯＢＯＬ　ＤＭＬプログラムの翻訳から実行までの時間を含まない）を測定する。

また端末に対して検索結果を出力するのに要する経過時間を測定し、通信コストとして１メッセージに要する時間を評価する。

b）最適化の評価

生成されるＣＯＢＯＬ　ＤＭＬプログラムの応答時間を短縮する為に、問合せ変換内で最適化を行なっている。ここでは、最適化処理の性能及び生成されたＣＯＢＯＬ　ＤＭＬの性能を評価する。

①　最適化の仮定の検証及び最適化のｃｐｕ負荷とその効果

「検索の応答時間は、アクセスするオカーランス数に比例する」という最適化の仮定に対して検証を行う。

また、同一問合せに対して最適化でサーチするレベルの深さ（ＤＦＡ１、ＤＦＡ２、ＤＦＡ３）による最適化のｃｐｕ負荷と、各最適化によって生成されたＣＯＢＯＬ　ＤＭＬ

プログラムの応答時間によってその効果を比較する。

② 選択度の評価

最適化でのコスト見積りで使用している選択度と実際にデータベースをアクセスして満足した割合とを比較する。

③ 生成されたＣＯＢＯＬ　ＤＭＬプログラムの性能評価

ＣＯＢＯＬ　ＤＭＬプログラム内を次の３つに分けて、各処理の経過時間を測定する。

- 初期／終了処理
- ｎａｖｉｇａｔｉｏｎ処理
- 出力処理

ｎａｖｉｇａｔｉｏｎ処理とは、オカーランスを検索する（ｆｉｎｄ）ための処理である。出力処理とは、検索結果を出力するための処理である。

c）格納構造とアクセス効率の評価

一般に、データベースは格納構造によって、そのアクセス効率が異なる。例えば、セット型の親レコードの近傍（物理的な近傍）にその子レコードオカーランスを格納するクラスタソングでデータベースにオカーランスを格納することがある。この場合、セット型の同一セットオカーランス内のオカーランスへのアクセスとｐａｇｉｎｇが減少する。

ここでは、実際にデータベースの格納構造をクラスタリングした場合とクラスタリングしない（ノンクラスタリング）場合でデータベースを作成する。同一問合せを各々のデータベースで実行させ、アクセス効率を評価する。

d）ＣＯＤＡＳＹＬ型ＤＢＭＳ間の互換性の達成

ＣＯＤＡＳＹＬ型の種々のＤＢＭＳに対して、共通インタフェースとしてＬＤＰ－Ｖ１.５が同程度の動作をすることと、有効な性能をもつことを検証する。

Ｂ　方　　針

評価の方針は、以下の様である。

a）単一ユーザのもとでＬＤＰ－Ｖ１.５を実行する。

他のユーザジョブによる影響の無い状態での性能を評価する。

b）ＬＤＰ－Ｖ１.５の各モヂュール及び生成されたＣＯＢＯＬ　ＤＭＬプログラムの各サブルーチンに対して、各々ＰＬ／ＩとＣＯＢＯＬが提供する時間取得（ｅ．ｇ．　ＣＯＢＯＬでは、ＡＣＣＥＰＴ　ＴＭ　ＦＲＯＭ　ＴＩＭＥ、ＰＬ／ＩではＴＭ＝ＴＩＭＥ）によって、その経過時間を測定する。

c）各問合せに対して、実行は１回行なう。Ｍ－１７０Ｆで経過時間を単一ユーザ下で測定しても、１～2割程度の誤差が発生する。しかし、ここでは１回の実行における経過時間を測定する。

d）ＬＤＰ－Ｖ１.５は、Ｍ－１７０Ｆで実行させる。

e）ＬＤＰ－Ｖ１.５で生成されるＣＯＢＯＬ　ＤＭＬプログラムは各々ＡＩＭとＡＤＢＳ上で

実行させる。ＤＭＬⅠ（ＤＭＬパターン情報）の入れ換えで、各々に対応するＣＯＢＯＬＤＭＬプログラムを生成する。

f）格納構造によるアクセス効率の評価はＡＤＢＳで行なう。ＡＩＭでは、レコードオカーランスと関連性のローディングを別々に行なっている為、クラスタリングによる格納を行うにはローディングプログラムを作り直さなければならない。

C　評価用問合せ

ＬＤＰ－Ｖ１.５では、ＱＵＥＬ問合せを入力して、内部では、問合せグラフとして（ＬＱＧ、ＣＱＧ）表現して処理を行っている。グラフ表現を処理しているので、グラフ表現の要素である節点数や辺数が処理に要する時間を決定する要因と考えられる。そこで評価用問合せとしては、テスト番号が大きい程、節点数や辺数が多くなる様に作成した。また、内部のグラフ表現としては、ループを含む問合せグラフとループを含まない問合せグラフがある。ループを含む場合、合流節点が生じ、同じレコードオカーランスを複数回アクセスすることが必要となる。この為、評価問合せは付記Ⅱの様に作成した。

D　性能評価

a）Ｐｅｒｆｏｒｍａｒｃｅ　ｂｏｔｔｌｅｎｅｃｋの明確化とその改良及び効果

ＬＤＰ－Ｖ１.５の各モヂュール（ＲＥＴＲＶＰ、ＬＱＧＰ、ＣＱＧＰ、ＡＰＧ、ＤＭＬＧ）の経過時間を測定し、Ｐｅｒｆｏｒｍａｎｃｅ　ｂｏｔｔｌｅｎｅｃｋを明確にする。

(1)　各モヂュールの経過時間

各問合せに対する各モヂュールの経過時間は、図３.６２の様になった。問合せの番号が大きくなるにつれて生成されるアクセス木の節点数が多くなっている。

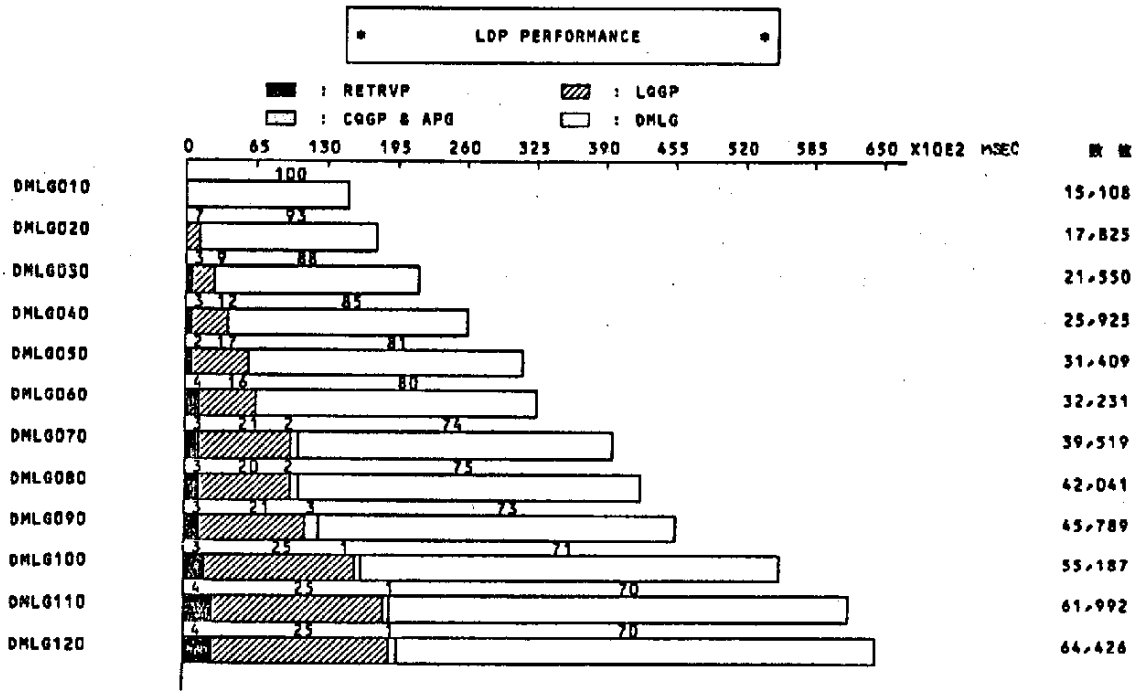


図３.６２　ＬＤＰ－Ｖ１.５

図からわかるようにモヂュールDMLGの経過時間は、全体の７０～９７％を占めている。即ち、モヂュールDMLGがLDP－V 1.5のｂｏｔｔｌｅｎｅｃｋとなっていることがわかる。

(2) DMLGの改良

モヂュールDMLGでは、以下の処理を行っている。

- DMLパターンの変換
- 変換の為のサーチ

DMLGでは、アクセス木の節点に対応するレコードオカーランスをアクセスするためのDMLパターンを生成する。DMLパターンは、レコード型名やセット型名等をパラメータとして含んだCOBOLステートメントから成っている。この為、パラメータをアクセス木の節点に対応するレコード型名、枝に対応するセット型名に変換する必要がある。変換の為のサーチでは、ステートメントの左から１文字ずつ読み込みパターンマッチングを行っている。

DMLG処理の問題点としては、次の点がある。変換の必要のない（パラメータを含まない）ステートメントに対しても、変換の必要があるステートメントと同様の処理を行っている。又、変換の為のサーチをステートメントの最後になるまで行っている。

以上より、次の様な改良を行う。

(i) "¥"と"%"（パラメータ）を含まないステートメント、即ち変換する必要のないステートメントと変換する必要のあるステートメントをｆｌａｇによって分離する。変換の必要のないステートメントはサーチを行なわない。

(ii) " @ "をステーメントの最後に付けることにより、変換の為のサーチ空間を小さくする。左から1文字ずつ読み込んで、" @ "を検出したら、変換処理を終了する。

図 3.6 3は、これらの改良後の結果を示している。この改良によって、改良前に比べて３～４割経過時間が改良されていることがわかる。

今後の改良としては、以下の点が考えている。

(i) コメント行の削除

DMLパターンの総ステートメント数1,2２０行内に、コメント行は２８２も含まれている。これらのコメント行を出力しない。

(ii) 徹底した左詰め

DMLパターンの各語間のスペースを減らし、サーチ空間を減少させる。

また、DMLGの改良の結果、モヂュールＬＱＧＰ経過時間が全体に占める割合が大きくなってきた。ＬＱＰの改良についても今後検討する必要がある。

Ｍ１６０からM－１７０Ｆへのマシンリプレースされたので、同一問合せを実行した結果が図 3.6 4である。図からわかるようにモヂュールＬＱＧＰ占める割合が大きくなっている。Ｍ１６０で実行させたＬＤＰ－Ｖ 1.5の改良版と比べると経過時間は半分くらい

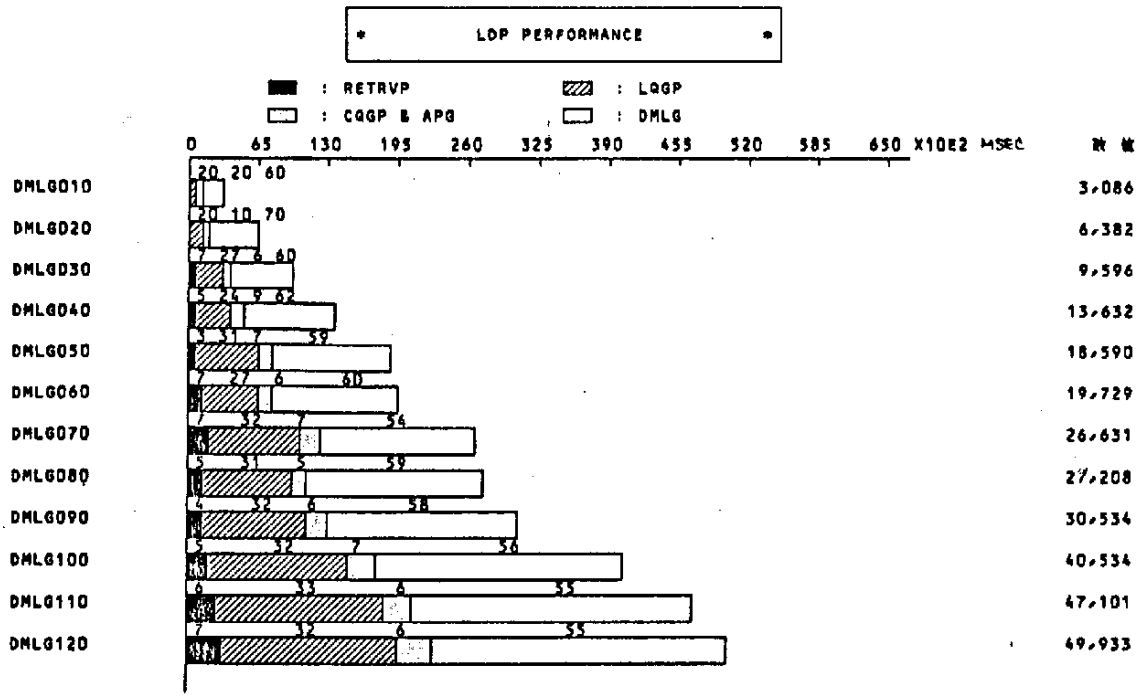


図3.6 3　LDP－V1.5の改良（M－160）

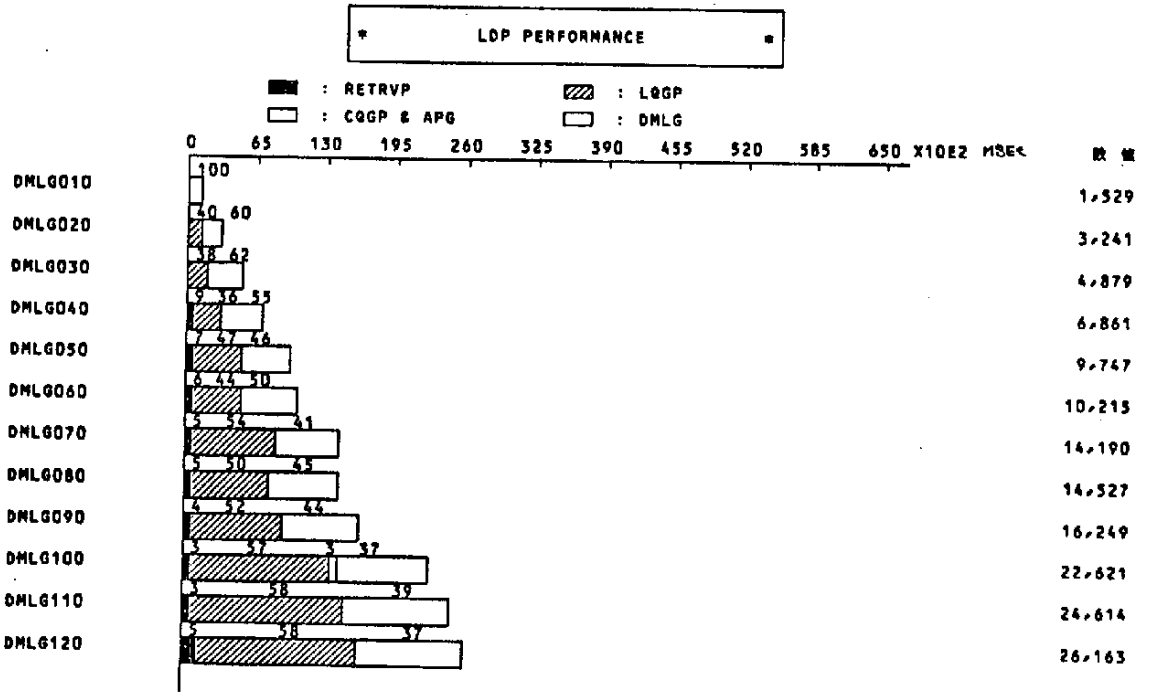


図3.6 4　LDP－V1.5の改良版（M－170F）

になっている。

b） ローカル通信処理（ＬＣＰ）の評価

ローカル通信処理は、ＩＴＰは用いてＣＯＢＯＬ　ＤＭＬプログラムの実行に必要なＪＣＬをリモートバッチシステム（ＲＥＳ）に送り、ＣＯＢＯＬプログラムを実行させる。また、検索結果も同様にＩＴＰ経由してＬＤＰ－Ｖ１.５に出力される。ここでは、ＬＣＰによる通信処理のオーバヘッドを測定する。ＬＣＰの通信処理概要は図3.６５の様である。

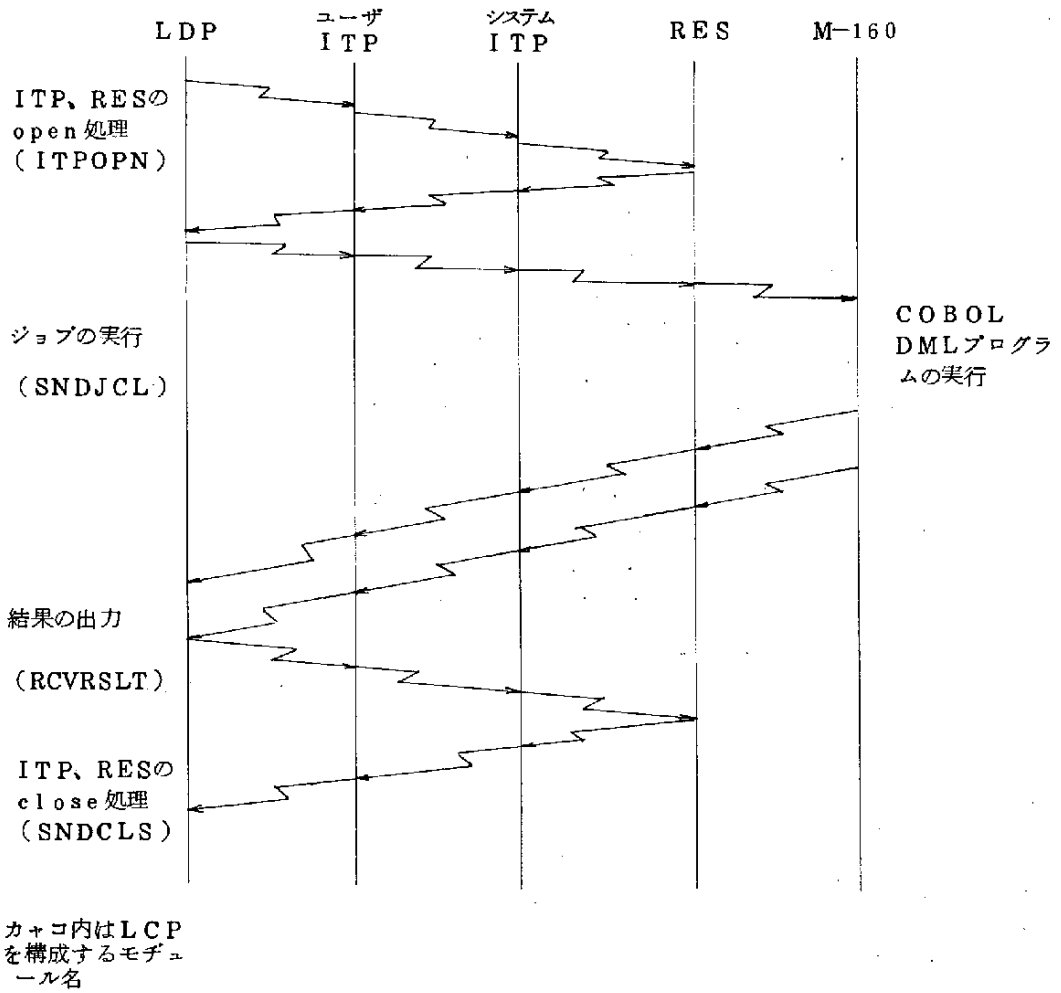


図 3.６５　ＬＣＰの通信処理概要

各通信ステップでの経過時間を測定し、ＬＣＰによる通信処理のオーバヘッドを明らかにする。また、端末への検索結果出力の経過時間を測定し、ＬＣＰの通信コストとしてメッセージ当りに必要な時間を測定する。

この評価では、評価データ収集の為にプリンタ端末からＬＤＰ－Ｖ１.５を使用し、ＬＣＰの性能評価を行った。問合せに対する各ステップの経過時間は、表3.２６の様である。

表 3.26　ＬＣＰの性能評価

問合せ	ＬＣＰＯ１０	ＬＣＰＯ２０	ＬＣＰＯ３０	ＬＣＰＯ４０
ＩＴＰＯＰＮ（ＩＴＰ、ＲＥＳのオープン）〔msec〕	16,822	15,701	17,817	17,061
ＳＮＤＪＣＬ（ジョブの実行）〔msec〕	76,810	77,037	76,405	148,579
ＲＣＶＲＳＬＴ（結果出力）〔msec〕	186,578	79,058	52,222	415,634
ＳＮＤＣＬＳ（ＩＴＰ、ＲＥＳのクローズ）〔msec〕	4,548	1,962	4,394	7,787
ＴＯＴＡＬ　ＴＩＭＥ〔msec〕	284,758	173,758	150,838	589,061
メッセージ数	82	79	42	309
バイト数	4,388	1,404	965	8,954

ＣＯＢＯＬ　ＤＭＬプログラムを実行させるために必要な通信処理のオーバヘッドとしては、ＬＣＰ内のモデュール（ＩＴＰＯＰＮ、ＳＮＤＣＬＳ）の経過時間である。このオーバヘッドは、約２０～２７秒である。

出力速度の遅いタイプライタ端末で結果を出力した為、出力に経過時間はほとんど文字数に比例する結果になった。通信能力として、１メッセージを受信するのに要する時間を評価することがでもなかった。

また、ＣＯＢＯＬ　ＤＭＬプログラムを実行させている間（約３～５分）、端末はＬＣＰに占有されてしまう。即ち、ＣＯＢＯＬ　ＤＭＬプログラムの実行の終了を待っている状態のままになっている。この間、ユーザは端末を使用することができない。実用システムとして、ＬＤＰ－Ｖ１.５を考える上で、ローカル通信処理のオーバヘッドの大きさと、結果出力が終るまで端末が占有されることが問題となる。

しかし、Ｍ－１７０Ｆでサポートされるｉｐｆにより、ＰＬ／Ｉプログラムからｔｓｓコマンドがｃａｌｌできるようになった。これを使用することにより、問合せ変換のＤＭＬＧモデュールでＣＯＢＯＬ　ＤＭＬプログラム生成後に、ＣＯＢＯＬ　ＤＭＬプログラムを実行させることができる。実用化システムに対する検討について詳しくは、付記ＩのＲＩＳＣ ＯＳを参照。

Ｅ　最適化の評価

問合せ変換で生成されるＣＯＢＯＬ　ＤＭＬプログラムの応答時間を短縮する為に、最適化を行っている。ここでは、最適化に要する経過時間及び生成されたＣＯＢＯＬ　ＤＭＬプログラムの応答時間について評価を行う。

a）最適化の評価

(1) 最適化の仮定の検証

アクセスパス生成において、「ＣＯＢＯＬ　ＤＭＬプログラムの応答時間は、アクセスされるオカーランス数に比例する。」と仮定〔 3.3.5　アクセスパス生成 〕し、最適化を行なっている。

この仮定の正しさを調べる為に、ＣＯＢＯＬ　ＤＭＬプログラムを実行させ、実際にアクセスしたオカーランス数とそれに要した時間を測定した。縦軸にアクセスしたオカーランス数、横軸にアクセスに要した時間を図 3.6 7は示している。

図から、アクセスされたオカーランス数に対して、アクセスに要する時間は、単調に増加していることがわかる。即ち、我々がアクセスパス生成で仮定した最適化の仮定は妥当であることが言える。

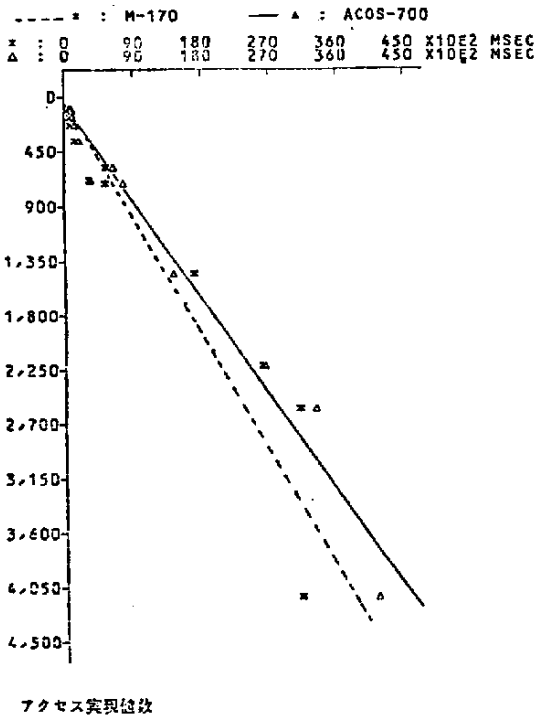


図 3.6 6　アクセス実現値数と応答時間

(2)　最適化の経過時間

アクセスパス生成アルゴリズムとしては、１段階のサーチを行うＤＦＡ１、２段階のサーチを行うＤＦＡ２と３段階のサーチを行うＤＦＡ３がある。ここでは、各マルゴリズムによる最適化の経過時間を測定する。

①　ＤＦＡ 1, 2, 3 の経過時間

アクセスパス生成アルゴリズムでは、ＣＱＧを入力して、アクセス木を生成している。各アルゴリズムによる最適化の経過時間を決める要素としてはＣＱＧの辺数が考えられ

る。ＤＦＡ１,２,３の各アルゴリズムによる最適化の経過時間は図３.６６の様である。縦軸はＣＱＧ内の辺数（セット数）で、横軸は各々の経過時間（ｍｓｅｃ）である。

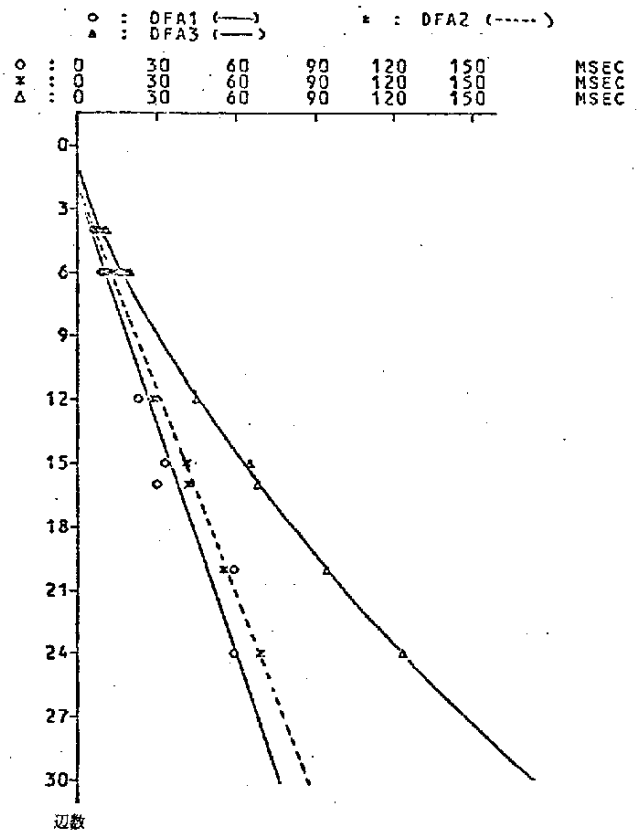


図３.６７　ＤＦＡ１,２,３の経過時間

この例では、ＤＦＡ１、ＤＦＡ２は、ほぼ同じ傾きの直線に近似（最少２乗法による）できる。これは、ＤＦＡ２で次の節点を見つける為にサーチする辺の数がＤＦＡ１に比べてほとんど増加してない為と考えられる。

また、ＤＦＡ３は二次係数が0.１、一次係数が３の二次曲線として近似できる。しかし辺数が３０程度までは、一次の係数が支配的で、ほぼ直線的に増加している。

(3)　最適化の効果

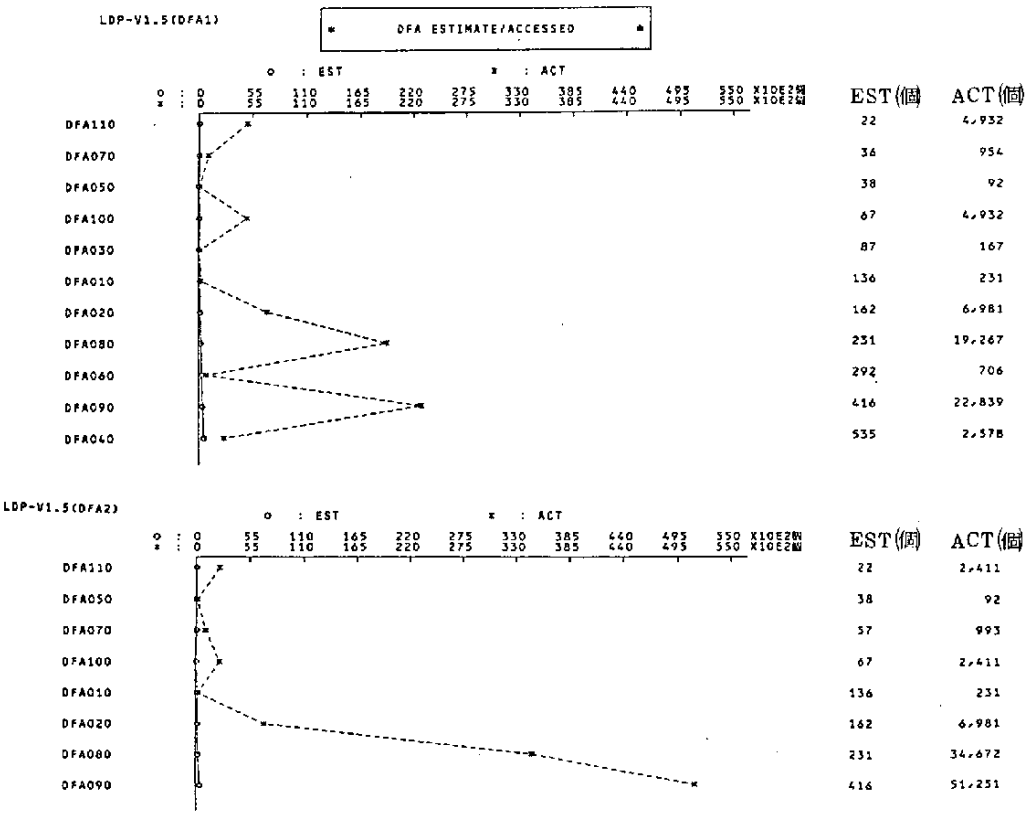
各評価関数ＤＦＡ１,２,３のもとで生成されたＣＯＢＯＬ　ＤＭＬプログラムのアクセス結果が表3.27である。表から、サーチレベルを深く見ることによる最適化の効果（アクセス結果の減少）は明らかでない。

表3.27 Estimate／Actual accessed occurrences by DFA1,2,3

Query #	No. of Node	No. of Join	DFA1		DFA2		DFA3	
			Est.	Act.	Est.	Act.	Est.	Act.
10	5	4	136	231	136	231	136	231
20	7	6	162	6,981	162	6,981	162	6,981
30	5	4	38	92	38	92	38	92
40	12	12	36	954	57	993	57	2,281
50	15	16	231	19,267	231	34,672	231	18,681
60	21	24	416	22,839	416	51,251	416	23,263
70	14	15	67	4,932	67	2,411	82	1,791
80	18	20	22	4,932	22	2,411	22	1,115

(4) 見積りとアクセス結果

アクセス見積り数と実際のアクセス結果とを比較すると、DFA 1,2,3に対して各々図



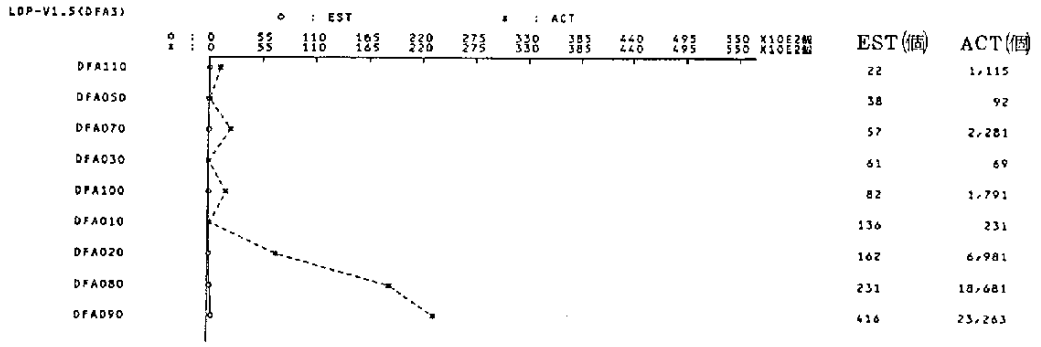


図3.68　見積りとアクセス結果

全体的に、見積り数が実際にアクセスされたオカーランス数よりも少くなっている。アクセスパス生成のコスト計算で使用しているパーフォマンス情報（結合度、選択度）は、データベース全体でオカーランスが均一に分散していることを仮定している。しかしこの例では、アクセスされた部分の値分布がこの仮定を満足していない為に見積りと一致していないと考えられる。

(5)　最適化評価の結論

最適化アルゴリズムDFA1、DFA2とDFA3による最適化の経過時間とその効果を評価した結果をまとめると次の様になる。

・最適化アルゴリズムDFA1、DFA2、DFA3による最適化の経過時間は、CQGの辺数が30程度まではほぼ直線的に増加する。また、最適化の経過時間の問合せ変換全体の経過時間に占める割合は数%にしか過ぎない。即ち、最適化は問合せ変換のオーバヘッドとなっていない。

・最適化の効果の面では、DFA1,2,3で生成されたCOBOL　DMLプログラムのアクセスオカーランス数の最適化が、レベルの深く見ることによって良くなっていることが明らかではない。

・アクセス見積り数と実際のアクセスオカーランス数に相互関連がない。

これらの点から、最適化では今後の改良として次の点を考える。

(i)　アクセス見積りの改良

(ii)　アクセスパス生成の改良

(i)　見積りの改良

見積りの値がアクセス結果と大きく異なる問題は、最適化で使用しているパフォーマンス情報(結合度、選択度)を値の分布が一様であることを仮定し、データベース全体の平均値を使用している事によると考えられる。

アクセス見積りの改良としては、アプリケーション毎にパフォーマンス情報を持つこ

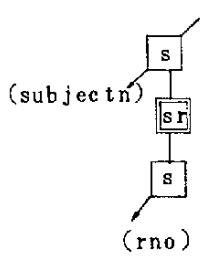
とと、問合せの実行結果をパフォーマンス情報に反映させる方法を考えられる。パフォーマンス情報のうち、選択度は値の分布により、個々の値に対してその選択度を考えることが必要になる。しかし、個々の値に選択度をもつことは一般には考えられない。ここでは結合度による見積りの改良を考える。

問合せの実行結果をパフォーマンス情報に反映させる方法としてｄｙｎａｍｉｃ ｔｕｎｉｎｇを考える。ｄｙｎａｍｉｃ ｔｕｎｉｎｇでは、実行された問合せが実際にアクセスした結合度（ｃｔ'（ｓ）と記す）とＨＩ内の結合度（ｃｔ（ｓ）と記す）とから次式により計算した値を新しい結合度とする。

$$ct(s) \leftarrow \frac{ct(s)+\alpha \cdot ct'(s)}{1+\alpha}$$

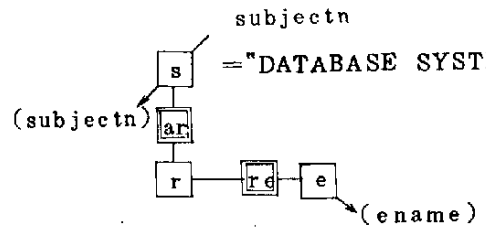
ここでαは重み付け係数である。

ここでは、αを１とし、ＨＩ内の結合度として最初はＤＢ全体の平均の結合度を持っているとする。このパフォーマンス情報に対して同じ問合せを繰り返し実行させた時のｄｙｎａｍｉｃ ｔｕｎｉｎｇによるアクセス見積りを測定する。

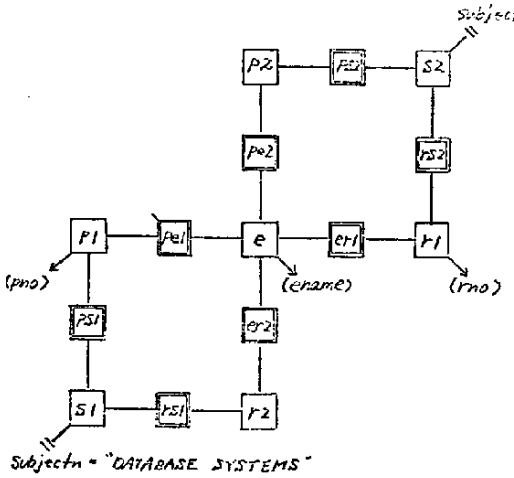


評価問合せ

	アクセス見積り数	アクセス結果
１回目の実行	１０	１１３
２回目の実行	６２	１１３
３回目の実行	８７	１１３
４回目の実行	１００	１１３
５回目の実行	１０５.６	１１３



	アクセス見積り数	アクセス結果
１回目の実行	２７.５	３３１
２回目の実行	１７７.９	３３１
３回目の実行	２５４.１	３３１
４回目の実行	２９２.５	３３１
５回目の実行	３１１.７	３３１



	アクセス見積り数	アクセス結果
1回目の実行	230	41,825
2回目の実行	3,035	41,825
3回目の実行	7,343	41,825
4回目の実行	10,892	41,825
5回目の実行	13,138	41,825

評価から、dynamic tuningによりアクセス見積り数がアクセス結果に近づいていることは明らかである。

しかし、問題は複数の異なる問合せが実行される環境でのdynamic tuningの効果である。

dynamic tuningで問題となるのは、アプリケーション内で実行される問合せがアクセスする結合度にばらつきが多い場合である。PRDBS内にアプリケーションを設定し、異なる複数の問合せを実行させdynamic tuningによるアクセス見積りの正しさを評価する必要がある。

また、問合せに対して生成されるアクセス木内に同じセット型に対応する異なる枝を持つ場合、アクセス木の下のレベルではアクセス範囲が限定されると考えられる。同じセット型としてこのアクセス結果を反映させ、dynamic tuningを行った場合のアクセス見積りの正しさを評価する必要がある。

dynamic tuningの今後の評価としては次の2点を考えている。

・アプリケーション内で複数の異なる問合せを実行させてdynamic tuningによるアクセス見積りを評価する。

・アクセス木内に同じセット型に対応する枝が複数ある場合のdynamic tuningによるアクセス見積りを評価する。

(ii) アクセスパス生成の改良

最適化の経過時間は、問合せ変換全体に対してオーバヘッドになっていないことからCQGをサーチするレベルをより深くすることが可能である。またアクセス見積り数が正しくなれば、アクセスオカーランス数の最適化がはかられると考えられる。アクセスパス生成の改良として次の事を考える。

問合せの節点数が少ない場合は、最適化においてアクセスパスを全面サーチして、最

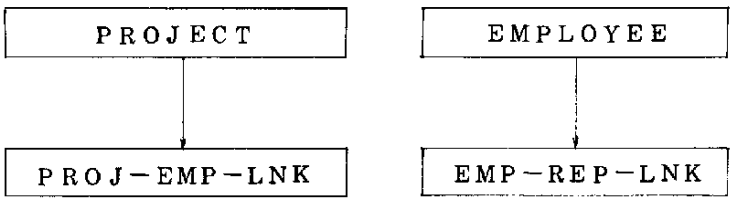
も最適なパスを見つける。また、問合せの節点数が多い場合は、１レベルのサーチアルゴリズムＤＦＡ１によりアクセスパスを生成する。

この為、最適化の入力であるＣＱＧの節点数、辺数による最適化の経過時間と生成されたＣＯＢＯＬ　ＤＭＬプログラムの応答時間に対する効果のｔｒａｄｅ－ｏｆｆ　ｐｏｉｎｔの明確化が必要となる。

b）選択度の評価

最適化におけるコスト見積りで使用する制限式に対する選択度と、実際にデータベースを検索して条件を満足したオカーランスの割合とを比較する。

実際にデータベースをアクセスした時に、条件を判定したオカーランス数と条件を満足したオカーランス数をプログラム内でカウントするようなカウント命令をＤＭＬ情報に組み込んで実行させた。選択度の評価では、次の様なセットに対して、子レコード型に各種の制限式を与えて実行させた。



(1)　次の制限式に対する選択度と実際に条件を満足した割合とを比較した。

制限式

$$\begin{cases} x.\ a = v \\ x.\ a \neq v \end{cases}$$

実際に比較した結果は表3.２８の様である。属性ＳＹＥＡＲは、値の分布がランダムに分布している属性で、値が８１にピークを持っている。また属性ＡＵＴＨ－ＮＯは、値が最小値（＝１）を持つオカーランス数が一番多く、最大値（＝５）に行くに従って急にオカーランス数が少なくなくなっている。

表からわかるように、選択度の計算値はほとんどの場合、実際の選択度に近似している。値の分布で大きくはずれる部分が計算値と大きく異なっている。

表3.28　選択度の評価

vの値	選択度（計算値）	実際の選択度
73	0.125	0.175
74	0.125	0.024
75	0.125	0.046
76	0.125	0
77	0.125	0.085
78	0.125	0.115
79	0.125	0.180
80	0.125	0.080
81	0.125	0.437

PROI－EMP－LNKの属性SYEAR
に対する制限式x、a＝vの比較表

vの値	選択度（計算値）	実際の選択度
73	0.875	0.890
74	0.875	0.984
75	0.875	0.968
76	0.875	1.000
77	0.875	0.997
78	0.875	0.997
79	0.875	0.888
80	0.875	0.991
81	0.875	0.552

PROJ－EMP－LNKの属性SYEAR
に対する制限式x、a≠vの比較

vの値	選択度（計算値）	実際の選択度
1	0.2	0.695
2	0.2	0.258
3	0.2	0.089
4	0.2	0.05
5	0.2	0.02

EMP－REP－LNKの属性AUTH－NO
に対する制限式x、a＝vの比較

vの値	選択度（計算値）	実際の選択度
1	0.8	0.496
2	0.8	0.666
3	0.8	0.911
4	0.8	0.944
5	0.8	0.977

EMP－REP－LNKの属性AUTH－NO
に対する制限式x、a≠vの比較

(2)　制限式　x．a $\left\{\begin{array}{l}\geq\\ \leq\\ >\\ <\end{array}\right\}$ vに対する選択度の評価

x．a $\left\{\begin{array}{l}\geq\\ \leq\end{array}\right\}$ v　に対する選択度は、

$(1-st_{xa})/n+st_{xa}$

x、a $\left\{\begin{array}{l}>\\ <\end{array}\right\}$ v　に対する選択度は、

$(1-st_{xa})/n$

である。但し、st_{xa}はx＝aの選択度である。

上記の選択度に対する式で、n＝2,3,4の場合についての計算値と実際の選択度について比較する。

比較結果は、図3.69の様である。図からわかるようにほとんどの場合において、n＝2による選択度が実際の選択度に近い。

属性値に対するオカーランス数の分布でかたよりがある場合は、オカーランス数の分布の少ない方に対する制限式に対してはn＝4の方が近似する（属性ＳＹＥＡＲに対する、制限式ＳＹＥＡＲ　ＬＥ　７５やＳＹＥＡＲ　ＬＴ　７５の場合）。

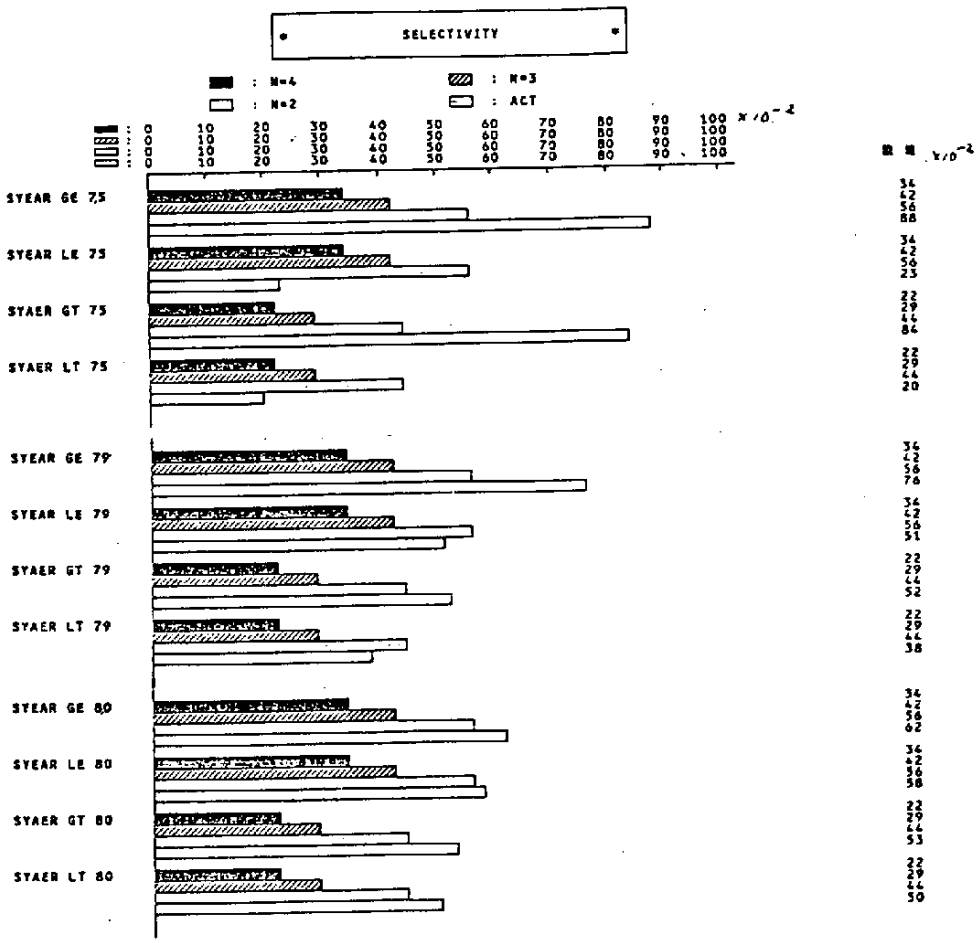


図3.69　選択度の評価結果

(3)　選択度評価のまとめ

選択度は、値によるオカーランスの分布が一様でない場合（特に、特定の値に集中してオカーランスがある場合）は、実際に条件を満足する割合に近似しない。アプリケーションが使用する値の範囲で、オカーランスの分布が一様な場合、選択度をアプリケーション

が使用する範囲内の平均値とすることによって選択度が実際の値に近づくと考えられる。

c) 生成されたＣＯＢＯＬ　ＤＭＬプログラムの評価

ＣＯＢＯＬ　ＤＭＬプログラムを次の３つ実行ステップに分けて考える。

(1) 初期／終了処理

(2) navigation処理

(3) 出力処理

(1) 初期／終了処理

生成されるＣＯＢＯＬ　ＤＭＬプログラム内で、初期化や終了処理を行なっている処理に対する経過時間を測定した。経過時間はＡＩＭ、ＡＤＢＳで各問合せに対してほぼ一定の値を示した。

ＡＩＭ　　1.3～1.6秒

ＡＤＢＳ　　5.8～7.0秒

ＡＤＢＳでは、各エリアがファイルと１対１に対応しているため、ＡＩＭに比べてファイルのＯＰＥＮ／ＣＬＯＳＥが多くなっている。この為、ＡＩＭに比べてＡＤＢＳの方が経過時間がよけいにかかっている。

アクセスするオカーランス数が少ない場合（数個～数十個）は、ＣＯＢＯＬ　ＤＭＬプログラムの応答時間に初期／終了処理の経過時間の占める割合は大き　なる。

(2) ｎａｖｉｇａｔｉｏｎ処理

ｎａｖｉｇａｔｉｏｎ処理とは、アクセスパス生成で生成されたパスをたどってレコードオカーランスをアクセスする処理である。ｎａｖｉｇａｔｉｏｎ処理は、アクセスしたオカーランス数の増加に対してほぼ直線的に経過時間が増加している。またアクセスするオカーランス数が多くなるにつれて、ＣＯＢＯＬ　ＤＭＬプログラムの応答時間に占める割合は大きくなる。

アクセス オカーランス数	全体に対する ｎａｖｉｇａｔｉｏｎ処理の割合
１００～1,０００	３５～７５％
1,０００～2,０００	７５～９０％
2,０００～5,０００	９０～９５％

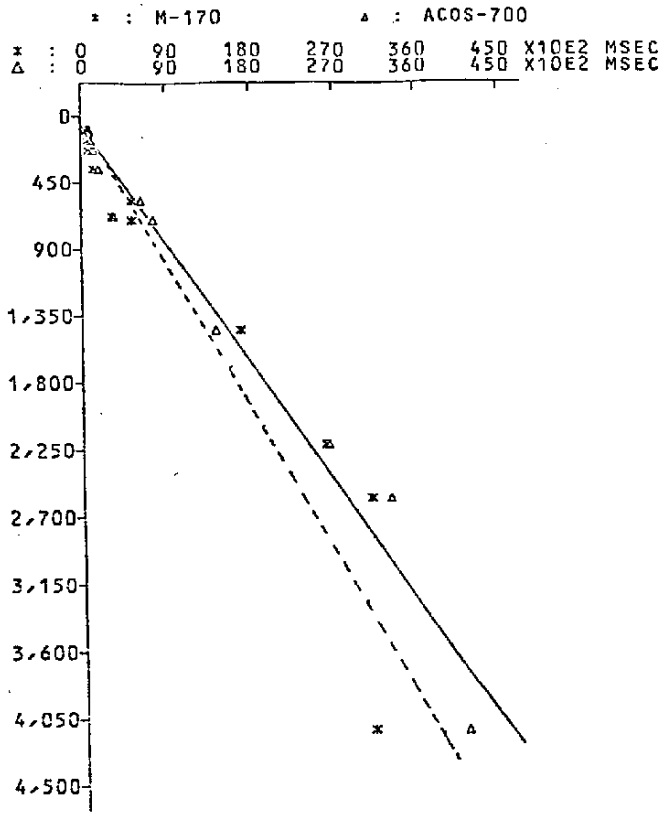


図 3.7 0　アクセス実現値数と応答時間

(3)　出力処理

実行した問合せに対して出力オカーランス数は図 3.7 1の様である。図から、出力オカーランス数と出力処理の経過時間との関係性は明らかではない。

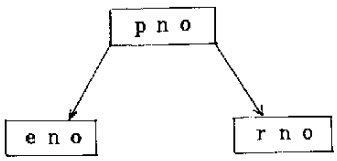
出力処理の全体に占める割合は１０％前後となっている。

出力処理の経過時間を決める要因としては、次のものが考えられる。

・出力属性数

・出力属性間の従属性

ここで、出力属性間の従属性を説明する為に次の様な出力属性の関係があるとする。



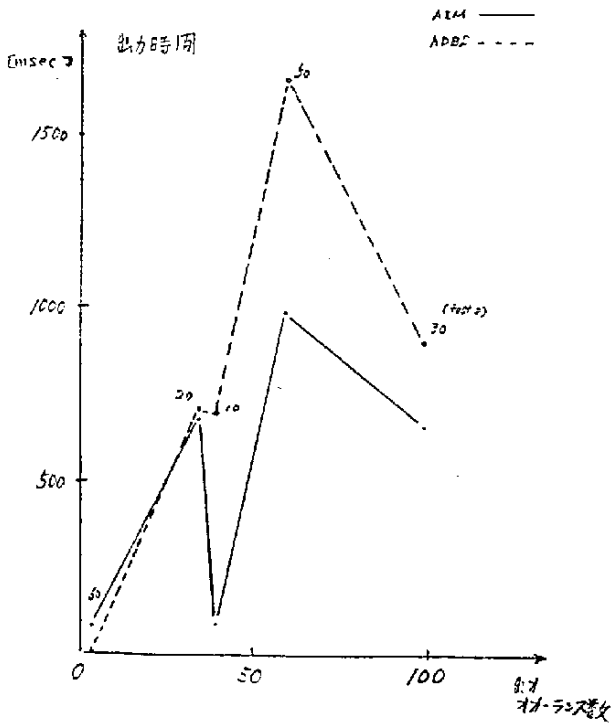


図3.7 1　出力数と出力時間

いま、出力属性として、ｅｎｏ（従業員番号）、ｐｎｏ（プロジェクト番号）、ｒｎｏ（レポート番号）があるとする。

P_1 というレコードオカーランスに対して、各セットオカーランスが図3.7 2の様になっているとする。

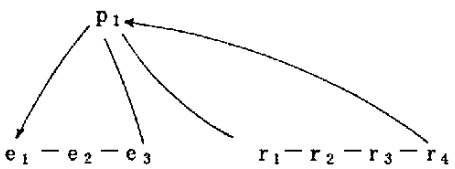


図3.7 2　p_1 に対するセットオカーランス

この時、問合せに対して、次の様な2つのアクセスパスを考える。

① ｐｎｏからｅｎｏ、ｒｎｏを各々アクセスする。

即ち　ｐｎｏ →→ ｅｎｏ
　　　　→→ ｒｎｏ

（→→セットを介した従属性を示す。）

② ｅｎｏからｐｎｏを経由してｒｎｏをアクセスする。

即ち ｅｎｏ→→ｐｎｏ→→ｒｎｏ

上記のアクセスパスから出力される結果は次の様になる。

①	p_1	e_1	r_1
		e_2	r_2
		e_3	r_3
			r_4
②	e_1	p_1	r_1
			r_2
			r_3
			r_4
	e_2	p_1	r_1
			r_2
			r_3
			r_4
	e_3	p_1	r_1
			r_2
			r_3
			r_4

上の様に異なったアクセスパスを取ることにより出力される量が大きく変化することが考えられる。現在、最適化として、アクセスするオカーランス数の最少化は行っているが、出力数の最適化の手法を今後検討する必要がある。

F 格納構造の評価

クラスタリングを行なうためには、各セットオカーランスが必らず同一ページに格納されなければならない。各セットオカーランスが１ページ内に格納されるために、ページサイズと格納比率を決める必要がある。

まづ最初に、各セットに対してセットオカーランスの最大長を求める。

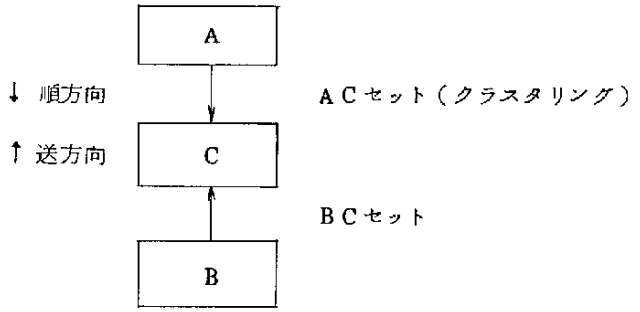
セットオカーランスの最大長＝親レコードオカーランス長
＋最大メンバ数×子レコードオカーランス長

セットオカーランスが１ページ内に格納されるように、ページサイズが次の条件を満たすように決められる。

ページサイズ×格納比率≧セットオカーランスの最大長

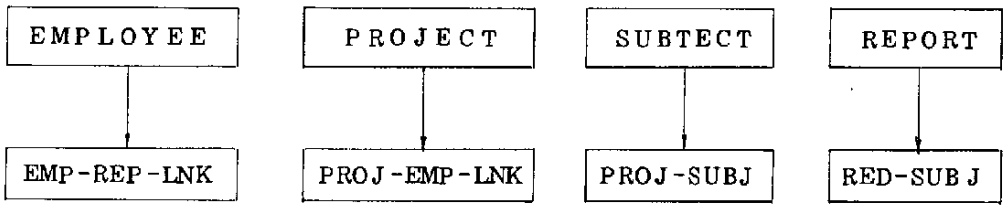
（但し、格納比率を８０％とした。）

以下の問合せに（ＣＱＧ）について評価する。



クラスタリングされているセットからアクセスするのを順方向とし、逆のセットからアクセスするのを逆方向と定義する。上の図の場合ＡＣセットからアクセスするのが順方向で、ＢＣセットからアクセスするのが逆方向である。

クラスタリングは、次の４つのセットに対して行なった。



順方向と逆方向についてのＣＯＢＯＬ　ＤＭＬプログラムの実行結果をまとめると表３.２９の様になる。

表3.29　実　行　結　果

順方向のアクセスタイム〔ｍｓｅｃ〕	
ノンクラスタリング	クラスタリング
5,850	7,420
960	830
3,710	5,930
17,990	18,230

逆方向のアクセスタイム〔ｍｓｅｃ〕	
ノンクラスタリング	クラスタリング
4,190	7,020
1,710	1,680
5,390	5,650
27,430	20,970

表からわかる様に、クラスタリングによってアクセスのｅｌａｐｓｅ　ｔｉｍｅ　が悪くなっている。ＣＯＢＯＬ　ＤＭＬプログラムの実行時にアクセスしたページ数を調べると表3.30の様になっている。

表3.30　アクセスページ数

順方向	
クラスタリング	ノンクラスタリング
280	304
18	11
49	142
1,029	1,101

逆方向	
クラスタリング	ノンクラスタリング
168	205
39	38
354	362
1,481	1,087

アクセスの経過時間とアクセスされたページ数を対応させると、クラスタリングによってアクセスしたページ数が増えていることがわかる。

この為にアクセスのｅｌａｐｓｅ　ｔｉｍｅが悪くなっていると考えられる。

評価用データベースＰＲＤＢＳでは、セット型に対して子レコードオカーランス数は平均3～4個と少ない。また、ネットワーク構造が多く、親レコードのアクセスからその子レコードを順次にアクセスする場合が少ない。

クラスタリングによるアクセス効率を評価するためには、次の条件が必要である。

・セットオカーランスが多数の子レコードオカーランスをもつ。

・親レコードのアクセスからその子レコードを順次にアクセスするアクセスパスで評価する

これらの条件を満たすデータベースでアクセス効率を評価することが必要である。

G　ＣＯＤＡＳＹＬ型ＤＢＭＳ間の互換性の達成

ＬＤＰ－Ｖ1.5で生成したＣＯＢＯＬ　ＤＭＬプログラムを各々ＡＩＭ、ＡＤＢＳで実行させた。各ＣＯＢＯＬ　ＤＭＬプログラムの実行に対して応答時間を比較すると表3.31の様になる。

表3.31　ＡＩＭ（Ｍ－１６０）とＡＤＢＳ（ＡＣＯＳ－７００）の応答時間

アクセスオカーランス数	92	168	231	707	2,578	4,932	6,981
ＡＩＭ　（ｍｓｅｃ）	3,100	3,850	3,950	11,960	45,520	67,420	117,330
ＡＤＢＳ　（ｍｓｅｃ）	7,600	8,610	8,640	19,750	44,190	58,140	188,410

アクセスしたオカーランス数が百のオーダーでは、ほとんど初期／終了処理による違いだけしか異なっていない。

アクセスしたオカーランス数が多い場合、ＡＩＭとＡＤＢＳでは多少異なっていることがわかる。しかし、どちらの方が良いかは、明らかではない。

全般的に見ると、ＡＩＭとＡＤＢＳに対してほぼ同等な性能をもっている。

Ｈ　性能評価のまとめと今後の問題

性能評価では、問合せ変換の性能評価からボトルネックとなったモヂュール（ＤＭＬＧ）の改良を行ない、３～４割経過時間を短縮することが出来た。また最適化の評価では、ＨＩ内のパフォーマンス情報を各アプリケーション毎にもつことによって見積りをアクセス結果に近づけることができた。しかし、次の様な問題点が残されている。

a）問合せ変換のボトルネック

問合せ変換ボトルネックＤＭＬＧにおいて、ＤＭＣＧの改良によって経過時間が３～４割短縮できた。この結果として、モヂュールＬＱＧＰの経過時間が全体に占める割合が高くなって来た。ＬＱＧＰでは問合せを表わす木を入力して、非手続的なグラフ生成を行っている。

b）ＣＯＢＯＬ　ＤＭＬプログラムの実行方法

我々は、ＣＯＢＯＬ　ＤＭＬプログラムの実行方法としてＪｉｐｎｅｔの対話型プロトコル（ＩＴＰ）を利用したローカル通信処理を実現した。性能評価から、ネットワーク負荷が大きく、実用システムとしてはユーザに対して大きな負担となることがわかった。ＣＯＢＯＬ　ＤＭＬプログラムを効率的に実行させる為には、ＯＳによるプロセス間通信機能が必要である。Ｍ－１７０ＦのＦＡＣＯＭ　ＯＳＩＶ／Ｆ４では、ＴＳＳユーザプログラムからＴＳＳコマンド（ｉ．ｅ．ＳＵＢＭＩＴコマンド）が利用できる。この機能を利用したシステムを付記ＩＲＩＳＣＯＳで検討している。

c）最適化処理

最適化では、２つの問題があった。１つは、サーチするレベルの深さによりアクセスオカーランスの最適化の効果が明らかでないことである。他の問題は、アクセス見積り数が実際にアクセスしたオカーランス数と相関関係がないことである。

アクセスパス生成アルゴリズムの改良として次の事を考えている。

(i)　ＣＱＧの辺数が少ない場合は、アクセスパスの全面サーチを行う。

(ii)　ＣＱＧの辺数が多い場合は、１レベルのサーチ（ＤＦＡ１）によりアクセスパス生成を行う。

現在、アクセス見積りについては、コスト計算で使用しているパーフォーマンス情報（結合度、選択度）はデータベース全体の平均から、各業務で使用する範囲の平均としての結合度を用いる方法の検討を行っている。これにより、アクセス見積りを正確にできることが期待される。同じ問合せを繰り返し実行した場合の評価ではアクセス見積りがよくなることがわかった。アプリケーションを設定し、アプリケーション内で複数の異なる問合せを実行した場合の評価を今後行う。

d）物理格納構造によるアクセス効率の再評価

格納構造としてクラスタリングによるアクセス効率の評価を試みたが、その効果は明らかではなかった。評価用データベースＰＲＤＢＳでは、親レコードオカーランスから、リンク

レコードを介して、別の親レコードオカーランスをアクセスする様な場合が多い。この為、クラスタリングによるアクセス効率を測定するには適していなかった。

しかし、セットオカーランスとして多数子レコードオカーランスを持つ場合に対しては、クラスタリングの効果を再評価する必要がある。

3.5.3 利用面からの評価

A 目　的

CODASYL DBSの手続的インタフェース（i.e. COBOL DML）を用いるよりも、リレーショナルインタフェース（i.e. QUEL）を用いることによりユーザが得る利得（gain）を明らかにする。

B 方　針

同一問題に対して、COBOL DMLによるプログラム作成とLDP－V 1.5を利用したQUELによる問合せの作成とで次の点について比較を行う。

設計コスト	（時間）
コーディングコスト	（〃）
デバッグコスト	（〃）
COBOLプログラムの実行	（〃）

但し、LDP－V 1.5を利用したQUELについては、これにLDP－V 1.5の実行時間を加えて比較する。

a) COBOL DMLによるプログラム作成

COBOL DMLによるプログラム作成では、以下の点について測定する。

- 設計時間
- coding時間
- debug時間

（トータル時間とコンパイル回数）

- COBOL DMLプログラムの質
  - アクセスパス
  - findするレコードオカーランス数
  - 実行時のtotal time
  - プログラムステップ数

b) QUEL

QUELによる問合せを作成するテスト対象としては、プログラム経験の無い者とする。

QUELによる問合せ作成では、次の点について測定する。

- QUEL問合せ入力時間
- LDPのオーバヘッド

（経過時間）

・ＬＤＰによって生成されたＣＯＢＯＬ　ＤＭＬプログラムの質

－　アクセスパス

－　ｆｉｎｄするレコードオカーランス数

－　実行のｔｏｔａｌ　ｔｉｍｅ

－　プログラムステップ数

Ｃ　テスト問合せ

テスト問合せとしては、以下の５つの問合せについて行う。ＣＯＢＯＬプログラマとＱＵＥＬ問合せ作成者に対しては、各問合せに対するＬＱＧの図を与えて、各々ＣＯＢＯＬ　ＤＭＬプログラムの作成とＱＵＥＬ問合せの作成を行なわせた。

各テスト問合せの意味は、以下の様である。

Ｒ１.　各ｐｒｏｊｅｃｔのメンバの書いたレポートのなかで、このメンバが第１著者（ａｕｔｈ－ｎｏ＝１）であるレポートの番号（ｒｎｏ）とそのｐｒｏｊｅｃｔの番号（ｐｎｏ）とを求めよ。

Ｒ２.　project のテーマと同じテーマの論文を書いているproject にメンバの番号(eno)と、その論文番号（rno）を求めよ。

Ｒ３.　分散型データベースシステム（ｄｉｓｔｒｉｂｕｔｅｄ　ｄａｔａｂａｓｅ　ｓｙｓｔｅｍｓ）又は、ｃｏｍｐｕｔｅｒ　ｎｅｔｗｏｒｋｓ、又はｌｏｃａｌ　ｎｅｔｗｏｒｋｓ、又は、ｄａｔａｂａｓｅ　ｓｙｓｔｅｍｓ、又は、ｄａｔａ　ｍｏｄｅｌｓを研究しているｐｒｏｊｅｃｔのメンバの中で、Ｊｉｐｄｅｃに属し、かつ１９７０年以降に入社（ｈｉｒｅ－ｙｅａｒ）したメンバの名前（ｅｎａｍｅ）、番号（ｅｎｏ）及びそのｐｒｏｊｅｃｔ番号（ｐｎｏ）を求めよ。

Ｒ４.　ｄｉｓｔｒｉｂｕｔｅｄ　ｄａｔａｂａｓｅ　ｓｙｓｔｅｍｓと、ｄａｔａｂａｓｅ　ｓｙｓｔｅｍｓと、ｆｏｕｒｓｃｈｅｍａ　ｓｔｒｕｃｔｕｒｅと、ｑｕｅｒｙ　ｔｒａｎｓｌａｔｉｏｎを研究している。

ｐｒｏｊｅｃｔのメンバが書いたレポートの主題（ｓｕｂｊｅｃｔｎ）と、そのメンバの所属（ｏｒｇ－ｎａｍｅ）及び、メンバの番号（ｅｎｏ）を求めよ。

Ｒ５.　ｐｒｏｊｅｃｔのメンバの中で、ｐｒｏｊｅｃｔのテーマと同一のテーマの論文を書いているメンバの名前と、そのメンバが属している全てのｐｒｏｊｅｃｔの番号及び、このメンバが第１著者である全ての論文の番号（ｒｎｏ）を求めよ。

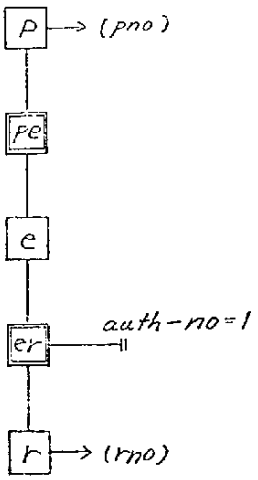


図3.7 3　R1のLQG

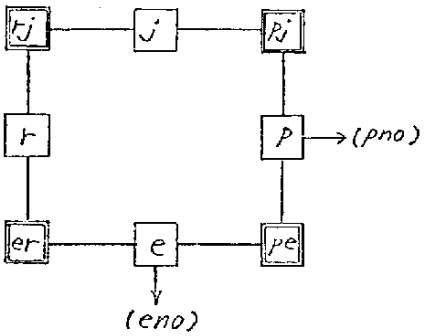


図3.7 4　R2のLQG

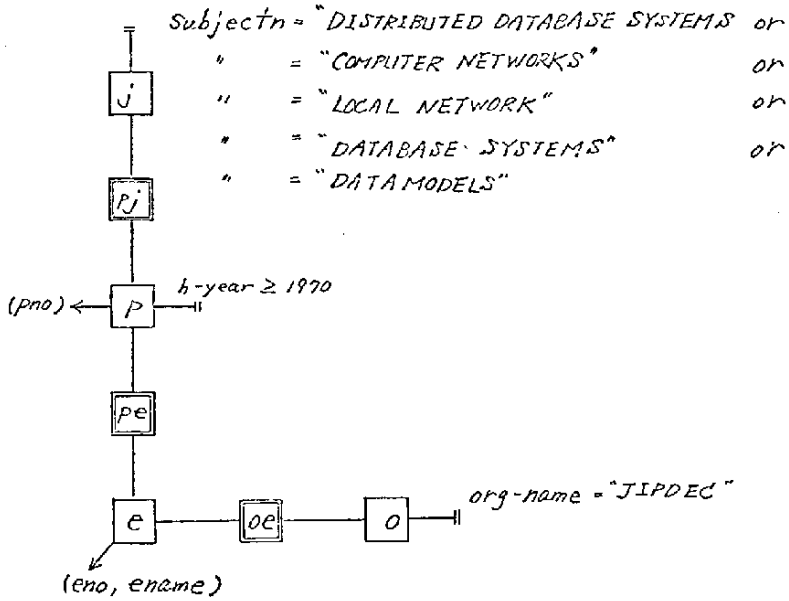


図3.7 5　R3のLQG

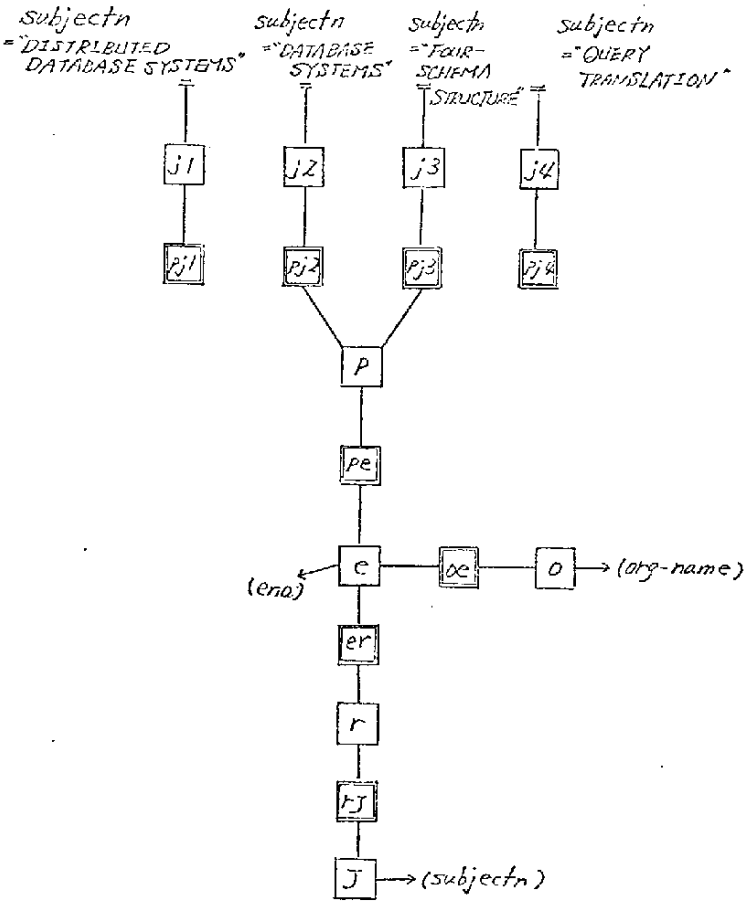


図3.7 6　R 4 のLQG

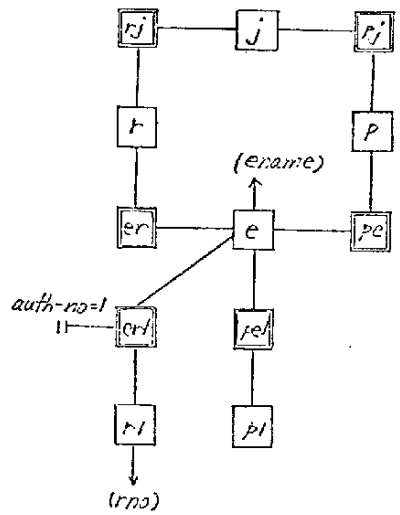


図3.7 7　R 5 のLQG

D ま と め

a) 全体コストの評価

COBOL DMLプログラミングによるコストとしては、設計、コーディング及びデバッグ時間を集計したものとする。又、QUEL問合せ作成によるコストとしては、QUEL問合せの入力時間及びLDP－V1.5の実行における経過時間を集計したものとする。

テストQueryに対して、実際にCOBOL DMLによるプログラミングとQUELによる問合せ作成を行なった結果が表3.32である。

表3.32 QUELとCOBOLのコスト比較

QUERY #	NO. OF RECORD TYPES	NO. OF SET TYPES		DESIGN CODING DEBUG	LDP ELAPSE TIME	NO. OF COBOL STEPS	NO. OF ACCESS OCCU.	COBOL ELAPSE TIME
R1	5	4	Q	4.5(M)	10(S)	461	335	4.5(S)
			C	10 (H)		104	335	3.5(S)
R2	8	8	Q	5.3(M)	16(S)	538	9748	140 (S)
			C	10 (H)		334	4150	65 (S)
R3	7	6	Q	11 (M)	12(S)	496	561	5.1(S)
			C	10 (H)		117	205	3.5(S)
R4	16	15	Q	14 (M)	22(S)	731	1831	27.7(S)
			C	17 (H)		256	1209	17 (S)
R5	12	12	Q	8 (M)	18(S)	653	59709	516 (S)
			C	15 (H)		284	1025	18 (S)

C=COBOL, Q=QUEL, (H)=HOUR, (M)=MINUTE, (S)=SECOND

QUEL問合せ作成の全体コストは、平均して十数分の単位であるのに対し、COBOL DMLでは十数時間の単位である。

これから明らかな様に、全体としては、QUELの方がCOBOL DMLによるプログラミングに対して高生産性を持つことがわかる。

b) 生成COBOL DMLプログラムの相違

直接COBOL DMLプログラムを作成した場合と、LDP－V1.5でQUEL問合せから生成されたCOBOL DMLプログラムを次の点について比較した結果が表3.31である。

・プログラムステップ数

・アクセスオカーランス数

・プログラムの実行時間

表3.3.2からわかるようにＱＵＥＬ問合せから生成されたＣＯＢＯＬ　ＤＭＬプログラムは、直接ＣＯＢＯＬ　ＤＭＬプログラムを作成したものに比べて約２～３倍程ステップ数が多くなっている。ＱＵＥＬ問合せから生成されるＣＯＢＯＬ　ＤＭＬプログラムは汎用的に作られている為、ステップ数が増大する。また、実際にＣＯＢＯＬ　ＤＭＬプログラムがアクセスしたオカーランス数は、人間の経験で選んだアクセスパスの方が良い結果を得ている。同じオカーランス数をアクセスしている場合（問合せＲ１の場合）でも、ＣＯＢＯＬ　ＤＭＬプログラムを直接作成した方が実行時間が短くなっている。

c) ＣＯＢＯＬ　ＤＭＬプログラミングの問題点

与えられた問合せに対して、ＣＯＢＯＬ　ＤＭＬによってプログラミングを行なった時に問題となった点は次の様である。

(1) コーディング量が多い

各問合せに対して、設計、コーディング、コンパイルテストの割合は図3.7.7の様になっている。これからわかる様に、平均して３０％～４０％もコーディングに時間を取られてしまう。

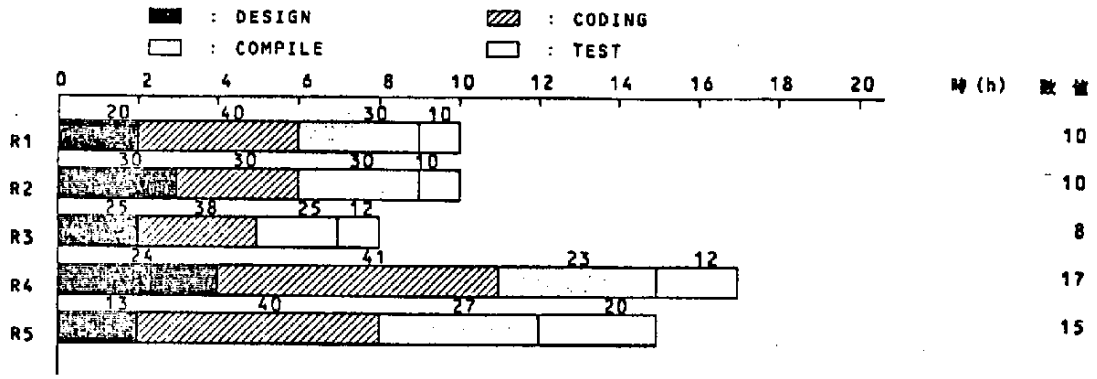


図3.7.8　プログラミングのコスト

(2) 現存子（ｃｕｒｒｅｎｃｙ）の概念

ＣＯＤＡＳＹＬデータベースで、ユーザにとって現在子（ｃｕｒｒｅｎｃｙ）である。問合せに対するプログラミングは、セット関係をたどってレコードオカーランスをアクセスし、条件を満足するレコードを見つける手続きを作成することになる。この時、複雑な問合せになると、条件を満足しなかった時にもどる位置の設定が難しい。

d) ＱＵＥＬの問題点

ＱＵＥＬ言語の問題点としては、ⅰ）グラフ表現が必要である、ⅱ）タイプミスが多く発生する点である。

ＱＵＥＬ問合せの作成過程を見てみると、与えられた問合せに対してＬＱＧに対応する問合せグラフをまづ作成し、その意味を確めてからＱＵＥＬ問合せを作成している。与えられた問合せから直接ＱＵＥＬ問合せを作成できるのは、節点数が数個の簡単な問合せだけである。

また、ＱＵＥＬ問合せの入力ミスでは、結合式の入力ミスが多い。これは、問合せのグラフ表現でループを含む様な場合、即ち同一節点から結合式がいくつもある場合に多い。

この様に、利用者とのインターフェースとしてＱＵＥＬ言語には問題点がある。ＱＵＥＬ問合せの作成過程の様に、問合せ入力を問合せグラフとして入力する方法の検討が必要となる。

e) ＱＵＥＬ問合せとＣＯＢＯＬの機能比較

ＬＤＰ－Ｖ1.5のＱＵＥＬは、ＣＯＢＯＬによるプログラミングに対して、次の機能をもっていない。

出力処理（出力の編集機能）

集積関数

統計関数

ＬＤＰ－Ｖ1.5では、ＱＵＥＬ問合せに対して問合せの結果を編集して出力する機能をもたない。出力処理に関しては、ＬＤＰ－Ｖ1.5で得られた検索結果に対して、出力の編集を行うインタフェースを設けることにより処理出来るものと考えている。

3.5.4 結　論

ＱＵＥＬ問合せからＣＯＢＯＬ　ＤＭＬプログラム生成までのＬＤＰ－Ｖ1.5の実行時間と応答時間は、数分の単位で終了する。また、利用者に対してＨＩ情報やＬＤＰ－Ｖ1.5のオブジェクトのリソースコスト（約５００KB）も問題となる様な大きさではない。

また、同一問題に対するＱＵＥＬによる問合せ生成と直接ＣＯＢＯＬ　ＤＭＬプログラムを生成することの比較によって、ＬＤＰ－Ｖ1.5のメリットとして次の点が明らかになった。

・ソフトウェアの高生産性

・非プログラマーであるユーザでも容易にデータベースが利用出来る。

これらのことから、ＬＤＰ－Ｖ1.5は非定型業務の検索利用に関してＣＯＤＡＳＹＬデータベースのインタフェースとして充分適用が可能である。

しかし、評価でも明らかにされた様に、最適化が人間の経験よりも劣る問題が残されている。最適化については、パーフォマンス情報を各業務によって使用される範囲で平均した値を使用することや、問合せのジョイン数によりサーチするレベルを変えることにより改良を計る。

ローカル通信機能については、ＩＰＦ（会話型プログラミング　ファシリティ）を利用し、オ

ーバヘッドの少ないＣＯＢＯＬプログラムの実行方式（付記Ｉ　ＲＩＳＣＯＳを参照）を検討する。

また、機能面では、出力処理、集積関数や統計関数が無い点が問題である。これに対しては、出力処理はＬＤＰ－Ｖ１.５の検索結果に対して、別のインタフェースとしてフォーム処理を行うことを検討している。また集積関数と統計関数については、ＬＤＰ－Ｖ１.５内で実現することを検討している。

３.６　まとめと今後の課題

本章では、ＣＯＤＡＳＹＬデータベースに対するリレーショナルインタフェース（ＬＤＰ－Ｖ1.5）の設計、実現及びその評価について論じた。ＬＤＰ－Ｖ１.５は、検索利用に限定し、リレーショナル問合せ（ＱＵＥＬ問合せ）を入力とし、ＣＯＢＯＬ　ＤＭＬプログラムを生成し、ＣＯＤＡＳＹＬ型ＤＢＳ（具体的にはＡＩＭ及びＡＤＢＳ）でこれを実行させ結果を得るものである。現在、実現を終了したＬＤＰ－Ｖ１.５は、次の機能を有している。

1）非手続的な検索ＱＵＥＬ問合せからＣＯＢＯＬ　ＤＭＬの生成

2）ＣＯＤＡＳＹＬデータベースシステム（ＡＩＭ、ＡＤＢＳ）での生成されたＣＯＢＯＬ　ＤＭＬプログラムの実働

3）結合として、主属性間の等価結合、不等価結合の処理

4）非主属性間の結合処理

5）ＬＣＳに対する階層的なビュー定義とビュー問合せ処理

6）ＣＯＤＡＳＹＬスキーマからリレーショナルスキーマの自動生成機能

7）ローカル通信機能（プロセス間通信）

実現されたＬＤＰ－Ｖ１.５に対して性能面と利用面から評価を行ない次の様な結論を得た。

1）ＬＤＰ－Ｖ１.５の性能として応答時間を考えると、問合せ変換プロセッサ内のＤＭＬ生成とＬＱＧ生成モヂュールがｂｏｔｔｌｅ　ｎｅｃｋとなっている。

2）ローカル通信処理は、システムの起動の煩しさやネットワークの負荷が大きく実用的でない。

3）最適化で見積ったアクセスオカーランス数が実際にアクセスした結果に一致しない。

4）アクセスパス生成で、最適化アルゴリズムとしてレベルを下げてサーチして見ても、アクセスオカーランスの最適化の効果が明らかでない。

5）利用面の評価では、ＬＤＰ－Ｖ１.５を利用したＱＵＥＬ問合せの方が、直接ＣＯＢＯＬ　ＤＭＬプログラムを作るより、生産性が高いことが明らかにされた。反面、人間の経験によるアクセスパス生成の方がＬＤＰ－Ｖ１.５の最適化よりも良いという結果を得た。

6）分散型データベースシステムにおける共通インタフェースとして、ＬＤＰ－Ｖ１.５は異種ＣＯＤＡＳＹＬ型ＤＢＳ（ＡＩＭ、ＡＤＢＳ）に対して生成されたＣＯＢＯＬ　ＤＭＬプログラムの応答時間はほとんど同じである。

上記の様に、ＬＤＰ－Ｖ１.５は当初の目的である分散型データベースの共通インタフェースとし

ての機能と、ＣＯＤＡＳＹＬデータベースシステムの一般ユーザインタフェースとしての機能を有していることを確認した。

しかし、評価からＬＤＰ－Ｖ１.５の今後解決していかなければならない問題としては、次の点がある。

1）アクセスパス最適化の方法

　アクセスオカーランスの最適化と出力量の最適化

2）ＣＯＢＯＬ　ＤＭＬプログラムの実行方法

3）出力結果の処理（フォーム処理）

4）統計処理機能

アクセスパスの最適化の方法として、アクセスされるオカーランス数だけでなく出力量も少なくする方法を検討する必要がある。また結合度をｄｙｎａｍｉｃ　ｔｕｎｉｎｇした時の最適化の効果を評価する必要がある。

ＣＯＢＯＬ　ＤＭＬプログラムの実行方法についてはＦＡＣＯＭ　ＯＳＩＶ／Ｆ４のＯＳ下のＡＩＭに対しては、付記ＩのＲＩＳＣＯＳによるリレーショナルインタフェースを検討している。出力処理については、ＬＤＰ－Ｖ１.５の上に別のインタフェースとしてのフォーム処理を、統計処理機能については、ＬＤＰ－Ｖ１.５の機能拡張による実現を考えている。

4. 更新機能を備えたリレーショナルインタフェースシステム（ＬＤＰ－Ｖ２）

第３章では、ＣＯＤＡＳＹＬデータベースシステムに対する検索用のリレーショナルインタフェースシステムＬＤＰ－Ｖ1.5についてその設計、実現及び評価について述べた。本章では、ＬＤＰ－Ｖ1.5に、ローカル概念スキーマ（ＬＣＳ）を通しての非手続的更新演算をＣＯＢＯＬ　ＤＭＬ　プログラムに変換し実行させる機能を付加したローカルデータベースプロセッサＬＤＰ－Ｖ２について述べる。ＬＤＰ－Ｖ２は、現在その実現を終了し、当協会のＭ－１７０ＦのＡＩＭ上で実働している。

本章では、4.1でＬＤＰ－Ｖ２の目的と概要を述べる。4.2では、ＣＯＤＡＳＹＬスキーマから更新用のローカル概念スキーマ（ＬＣＳ）の生成について論じる。4.3では、更新演算の変換について述べる。

4.1　ＬＤＰ－Ｖ２の目的と概要

ＣＯＤＡＳＹＬデータベースシステムは、ネットワーク型のデータ構造と、手続的なアクセス言語とをユーザに提供し、パフォーマンスを重視した定型的な大規模データベースに利用されている。この理由は、データ構造の物理的な側面と論理的側面とが未分離であることと、アクセス言語が手続的であることである。一方、この点は、エンドユーザのデータベース利用と、非定型的なデータベース利用ソフトウェア生産を困難にしてきている。又、各サイトにおいて独立に生成されてきた種々のデータベースシステム（ＤＢＳ）を、統合的に１つの巨大システムとして利用しようとする分散型データベースシステム（ＤＤＢＳ）においても、手続的インタフェースは、その運用面において問題がある。

上記の問題を解決する方法としては、ＣＯＤＡＳＹＬ　ＤＢＳ上に非手続的な利用者インタフェースを設けることがある。９）、１０）は、リレーショナルモデルインタフェースを設定することを試みている。既に、検索利用に対するリレーショナルインタフェースは、LDP—V1.5が実動している。現在の問題は、ＣＯＤＡＳＹＬデータベース上の更新をどの様に非手続的インタフェース上でモデル化するかである。即ちセット型のメンバシップクラスにともなった消去演算の波及と、追加演算に対する入力条件とがインタフェースのモデルとして表現出来ねばならない。我々は、リレーショナルモデルは、１つのリレーション内の組の追加、消去と値の変更とをサポートするのみで、リレーション間の更新インテグリティを規定していないと考えている。このために、ＬＣＳ（local conceptual schema）モデルを更新インテグリティを表わす構造をリレーショナルモデルに付加したものとして設定した。

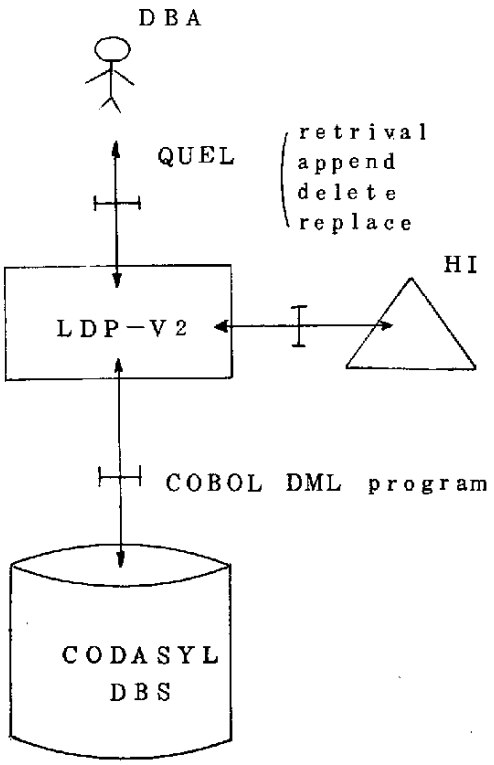


図4.1. LDP－V2の概要

4.2 スキーマ変換

本節では、ローカル内部スキーマ（ＬＩＳ）としてのＣＯＤＡＳＹＬスキーマから、ローカル概念スキーマ（ＬＣＳ）への変換について論じる。

リレーショナルモデルは、検索演算に対して高水準非手続的検索アクセスを許すモデルである。これに対して、ＬＣＳモデルは、このリレーショナルモデルに、更新の条件と伝播構造とを、モデル構造に取り込んだモデルである。従って、ＬＣＳに対する検索演算は、リレーショナルスキーマに対する検索演算と同じである。

4.2.1 変 換 目 標

スキーマ変換の目標としては、次の２点がある。

1. 情報の保存性
2. 更新の保存性

ここでＣＯＤＡＳＹＬスキーマをＣ、その外延をＣＯＤＡＳＹＬデータベースＣとする、ローカル概念スキーマ（ＬＣＳ）をＳ、その外延をＬＣＳデータベースＳとする。1.の情報の保存性とは、スキーマＣとＳとの外延の要素が互いに１対１に対応していることと定義する。

２は、更新演算に対して、この２つのスキーマが等価である条件を与えている。即ち、２つのモ

デルに於ける次の三点が互いに等しい時、２つのスキーマ間で更新保存性があると定義する。

1) 基本演算の更新対象

2) この演算が満足すべき更新条件

3) １）の基本演算によって生じる更新伝播

これは、図4.2の様に表せる。ここで、ＰＳとＰＣを、外延Ｓ及びＣに対する更新演算とする。Ｈを、ＣからＳを生成する関数とする。情報保存性を満足するためには、関数Ｈは、１対１対応でなければならない。この時、更新保存性は、以下が成り立つことである。

$$PS(S)=S' \qquad PC(C)=C' \qquad H(C)=S \qquad H^{-1}(S)=C$$

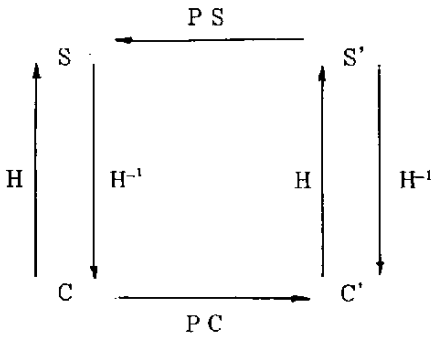


図 4.2　スキーマ変換

4.2.2 ＣＯＤＡＳＹＬモデル

ＣＯＤＡＳＹＬモデル〔ＣＯＤＡＳ７３〕は、事象の集合を表すレコード型と、事象間の関係性を表すセット型とから成るネットワークモデルの一種である。ＣＯＤＡＳＹＬモデルの第１の特徴は、リレーショナルモデルと異なり、データの論理的な構造に関する部分（概念スキーマレベル）と、データの物理構造に関する部分（内部スキーマレベル）とが未分離であることである。即ち、データ独立性が達成されていないことである。最近、前者をＤＤＬ78〔ＣＯＤＡＳ78ａ〕と、後者ＤＳＤＬ〔ＣＯＤＡＳ78ｂ〕とに、従来のＤＤＬ（データ定義言語）を分けることによって、データ独立性を達成することを試みているが、まだ十分ではない。多くの商用ＤＢＭＳも、ようやく〔ＣＯＤＡＳ７３〕を実現した（例、ＮＥＳのＡＤＢＳ　　）段階である。この為、本論文では、〔ＣＯＤＡＳ７３〕を、ＣＯＤＡＳＹＬモデルとして検討することにする。

ＣＯＤＡＳＹＬモデルの構造定義では、まず、モデルのなかをデータの論理構造に関する部分と、物理構造に関する部分とに分離し、各々について形式的定義を試みる。次に、ＣＯＤＡＳＹＬモデルのアクセス言語について論じる。

ここでは、ＣＯＤＡＳＹＬモデルのデータの論理構造について、その形式化を試みる。

ＣＯＤＡＳＹＬモデル（ここでは、以降、データの論理構造を表すＣＯＤＡＳＹＬモデルの部

分を意味するものとする）に基づいたスキーマ（ＣＯＤＡＳＹＬスキーマ）Ｓは、次の二つを基本構成要素としている。

i) レコード型　R

ii) セット型　S

これに対して、ＣＯＤＡＳＹＬスキーマＳの外延としてのＣＯＤＡＳＹＬデータベースＳは、次の二つを基本構成要素としている。

i) レコードオカーランス集合　R

ii) セットオカーランス集合　S

A　レコード型とレコードオカーランス集合

レコードオカーランス集合Ｒは、データベース内の事象の集合を表している。Ｄ０、Ｄ１、・・・、Ｄｎを値の集合とする。ここで、各Ｄｉは必ずしも互いに異なっている必要はない（ｉ≠０）。この時、レコードオカーランス集合Ｒは、次の様に定義される。

$$R \subseteq D0 \times D1 \times \cdots \times Dn$$

Ｒの要素ｒは、レコードオカーランスと呼ばれる。

ｒ＝（ｄ０，ｄ１，・・・，ｄｎ）　ｄ０∈Ｄ０，ｄｉ∈Ｄｉ（ｉ＝１，・・・，ｎ）。

レコードオカーランスｒ内の各値ｄｉの意味が、ｒ内のその位置に依存している制約を除く為に、各Ｄｉに対して、データ項目ｔｉを導入する。ｒのｔｉの値は、ｒ〔ｔｉ〕＝ｄｉ∈Ｄｉと表す。この時、Ｄｉをデータ項目ｔｉの定義域と呼び、ｄｏｍ（ｔｉ）＝Ｄｉと表わす。ＴをＲの全てのデータ項目の集合（ｉｔｅｍ（Ｒ）＝｛ｔｉｊ　｜　ｊ＝１，・・・，ｈｉ｝）のある部分集合とする。Ｔの値を次の様に定義する。

ｒ〔Ｔ〕＝（ｒ〔ｔｉ１〕，・・・，ｒ〔ｔｉｈｉ〕）

繰り返し句（ｒｅｐｅａｔｉｎｇ　ｇｒｏｕｐ）については、繰り返しの各々の要素を、独立したデータ項目として考えることにする。従って、繰り返し句は、明に考えないことにする。

Ｒの定義域の中で、Ｄ０は、その値としてデータベース内の各レコードオカーランスを一意に識別する値（これを、データベースキー又は、ｄｂキーの値と呼ぶ）から成る定義域とする。Ｄ０は、他のどのレコードオカーランス集合に対しても共通な定義域である。Ｒの各レコードオカーランスｒの中で、ユーザがデータ項目の値として結果を出力できるのは、ｔ０以外の部分である。

ＣＯＤＡＳＹＬモデルの特徴は、レコードオカーランス集合Ｒ内のレコードオカーランスｒを、一般にｔ０以外のデータ項目の値によって識別できない（ｎｏｔ　ｓｅｌｆ－ｉｄｅｎｔｉｆｉａｂｌｅ）事である。原則的に、レコードオカーランスｒは、Ｒ内での位置、即ち、ｔ０の値（ｄｂキー）によってのみ識別できる。即ち、

$$\forall r, r' \in R \quad r〔t0〕 \neq r'〔t0〕 \quad iff \quad r \neq r'$$

このｔ０を、ｄｂキーと呼ぶ。各レコードオカーランスに対して、そのｄｂキー値を求める関数を、特に、ｄｂ－ｋｅｙとする（即ち、ｄｂ－ｋｅｙ：ＲＲ→Ｄ０）。ここで、ＲＲはデ

ータベース内の全レコードオカーランスの集合である。この時、

$\forall r, r' \in RR \quad db\text{-}key(r) \neq db\text{-}key(r') \quad iff \quad r \neq r'$

$\forall r \in R \quad r[t0] = db\text{-}key(r)$

レコード型Rは、データ項目集合item(R)(={ti | i=1, ・・・, n})とインテグリティ条件IRとから成る。IRは、Rの外延Rの個々のレコードオカーランスが満足すべき条件を与えている。

$$R = \{ r \mid r \in D0 \times D1 \times \cdots \times Dn \wedge IR(r) \}$$

IRとしては、$\forall r \in R \quad r[t0] = db\text{-}key(r)$がある。レコード型Rは、次の様に表される。

$$\underline{R} \ (t1[/D1], \cdots, tn[/Dn])$$

B　セット型とセットオカーランス集合

セット型Sは、レコード型R1とR2との間に存在して、これらの間の1対Nの関係性を表し、S(R1、R2)と表される。R1とR2は、各々セット型Sの親及び子レコード型と呼ばれる。この時、R1=owner(S)，R2=member(S)と記す。

Sの外延としてのセットオカーランス集合Sは、レコードオカーランス集合R1とR2とに対する関係として定義される。

$$S \subseteq R1 \times R2 \quad (\text{ここで、} R1 \neq R2)$$

各Sに対して、以下の関数を定義する。

$\forall r1 \in R1 \quad fS(r1) = \{ r2 \mid (r1, r2) \in S \}$

$\forall r2 \in R2 \quad fS^{-1}(r2) = \{ r1 \mid (r1, r2) \in S \}$

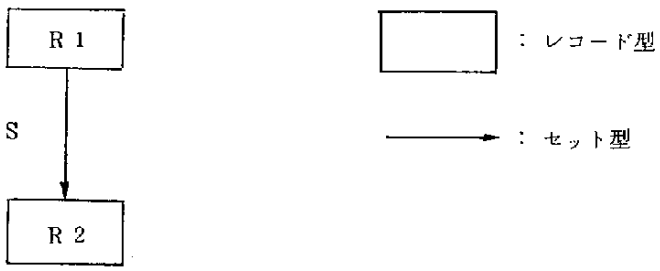


図4.　3セット型

fSは、Sのインテグリティ条件を与えている。この時、R1とR2とは、各々Sの親レコードオカーランス集合、子レコードオカーランス集合と呼ばれる。これは、CODASYLモデルでは、図4.3の様な図式によって表され、これをデータ構造図(Bachman diagram)呼ぶ。r2(∈R2)∈fS(r1)の時、r1をr2のSについての親レコードオカーランス、r2をr1のSについての子レコードオカーランスと呼ぶ。

セットオカーランス集合Sの特徴は、関数fSを用いて次の様に表される。

i) Sは、R1とR2との間の1対Nの関係性を表す。

$$\forall r1 \in R1 \quad card(fS(r1)) \geq 0$$

$$\forall r2 \in R2 \quad card(fS^{-1}(r1)) \leq 1$$

ここで、集合Aに対して、card(A)はそのカーディナリティである。

ii) R2の各レコードオカーランスは、Sについて、複数の親レコードオカーランスを持てない。

$$\forall r1, r1' \in R1 \quad if \quad fS(r1) \neq \phi \ \wedge \ fS(r1') \neq \phi,$$
$$fS(r1) \cap fS(r1') = \phi \quad iff \quad r1 \neq r1'$$

iii) R1のレコードオカーランスr1(∈R1)に対するSの子レコードオカーランスの集合は、順序づけられている。

$$\forall r1 \in R1 \quad fS(r1) = \{ r2i \in R2 \mid i = 1, \cdots, m \}$$

この時、次の順序リレーション に対して、r21 r22 ・・・ r2mである。tを、R2のあるデータ項目の集合({tk1, ・・・, tkh})とすると、

$$r2i \langle r2j \quad iff \quad r2i[t] \leqq r2j[t] \ \wedge \ i \theta j$$

ここで、

$$\theta = \begin{cases} \leqq & if \langle = ascending\ order \\ \geqq & otherwise(descending) \end{cases}$$

r2i[t]>r2j[t]の意味は、次の様になる。

$$r2i[t] \geqq r2j[t] = \begin{cases} true & \underline{if} \ r2i[tk1] > r2j[tk1] \\ & \vee (r2i[tk1] = r2j[tk1] \wedge \\ & \quad r2i[tk2] > r2j[tk2]) \\ & \vee \ \cdot \quad \cdot \quad \cdot \\ & \vee (r2i[tk1] = r2j[tk1] \wedge \\ & \quad r2i[tk2] = r2j[tk2] \\ & \quad \wedge \cdot \quad \cdot \quad \cdot \wedge \\ & \quad r2i[tkh] > r2j[tkh]) \\ false & \underline{otherwise} \end{cases}$$

これは、ソートにおいてtk1, ・・・, tkhが、左から右にmajor−minorの役目を持つことを示している。この時、データ項目の順序集合tをソート項目と呼び、sort−item(S)=tと記す。又、sort−type(S)=ソートのタイプ(∈{A(scending)、D(escending)})とする。sort−item(S)が在しない時、fS(r1)内のr2i∈R2は、DBMSが自動的に順序づける(e. g. 生成順)ものとする。

r1∈R1と、fS(r1)との集合は、特に、r1を親レコードオカーランスとするセットオカーランスと呼ばれる。これを、次の様に表す。

$$so(r1) = \{ r1, r21, r22, \cdots, r2m \}$$

ここで、　　r1＜r21＜r22＜・・・＜r2m

iv) r1∈R1のSに関する子レコードオカーランスに対して、これらのレコードオカーランス内で、あるデータ項目の値が一意で（DNA：DUPLICATES ARE NOT ALLOWED）であることが出来る。このデータ項目をtとする。

∀r1∈R1　∀r2, r2'∈fs(r1)　r2[t]≠r2'[t]

iff　　r2 ≠ r2'

この時、R2のデータ項目（集合）tを、セットオカーランス集合Sに関するDNAデータ項目（集合）と呼び、DNA－item（S）＝tと記す。

v) R1が、SYSTEMと呼ばれ、一つのレコードオカーランス（r1）から成るレコードオカーランス集合の時、Sを単項セットオカーランス集合と呼ぶ。

vi) Sは、R1とR2との間のインテグリティ条件を表わしている。これは、メンバシップクラスによって表される。メンバシップクラスには、格納クラスと消去クラスとがある。格納クラスには次の2つのオプションがある。

a) AUTOMATIC

b) MANUAL

R2にレコードオカーランスを格納する時、Sのあるセットオカーランスにも自動的に格納される場合が前者に相当する。この時、SのR2の格納クラスは、AUTOMATICであると言う。一方、格納クラスがMANUALである時、R2へのレコードオカーランスの格納と、このレコードオカーランスのSのセットオカーランスへの挿入とは、別の操作によって出来る。

消去クラスには、次の2つのオプションがある。

a) MANDATORY

b) OPTIONAL

Sの消去クラスが、MANDATORYである時、SのR2のレコードオカーランスは、これの属するセットオカーランスからこれが除去される時、同時に、これはR2からも消去される。一方、消去クラスがOPTIONALの時、レコードオカーランスのSのセットオカーランスからの除去は、これのR2からの消去を意味しない。

Sのメンバシップクラスとしては、次の2について考える事にする。

a) MANDATORY／AUTOMATIC（M／Aと記す）

b) OPTIONAL／MANUAL　　（O／Mと記す）

他の組合せ（MANDATORY／MANUAL、OPTIONAL／AUTOMATIC）は、実際あまり用いられないことから、本論文では上記2について検討することにする）。

a）のM／Aは、このSの子レコードオカーランス集合R2の全てのレコードオカーランスが、必ず、Sの親レコードオカーランスを持つ事を意味している。一方、O／MはR2の

レコードオカーランスは、必ずしも親レコードオカーランスを持たなくてよいことを表している。Sのメンバシップクラスが M／A であるとは関数 fS^{-1} が全関数、O／Mの時は半関数であることとして表せる〔表 4.1 〕。

即ち、

$\forall r \in R1$　　$card(fS^{-1}(r)) = 1$　　（M／A）

$\forall r \in R1$　　$card(fS^{-1}(r)) \leq 1$　　（O／M）

表4．1　メンバシップクラスと fS^{-1} のタイプ

メンバシップクラス	fS^{-1}
M／A	全関数
O／M	半関数

+）DDL78〔CODAS78a〕では更に、FIXEDもオプションに加えられている。
++）DDL78〔CODAS78a〕では、この他に（FIXED／OPTIONAL,FIXED／MANUAL）がある。
+++）ADBS〔ACOS700〕でも、これだけをサポートしている。

S（$\subseteq$ R1×R2 ）の要素 $s \in S$ は、次の様な2項組として表せる。

s（$\in$S）＝（r , r'）ここで、$r \in R1$, $r' \in R2$

sは、レコードオカーランスrがSに関してr'の親レコードオカーランスであり、r'はrの子レコードオカーランスであることを示している。

セット型Sのスキームは、次の様に表せる。

S （R1, R2）

C　CODASYLスキーマ

CODASYLモデルに基づいたスキーマは、ノードをレコード型、セット型をノード間のアークとする有向グラフとして表せる。これをSGとする。

SG＝（N, A）

N ＝ レコード型を表すノードの集合

A ＝ {（R1, R2）| $R1 \in N \wedge R2 \in N \wedge R1 \neq R2$}

ここで、アーク（R1, R2）はセット型Sを表すラベルSを持ち、owner(S)=R1かつmember(S)=R2であることを表す。CODSYLスキーマ対する制限として次の点がある。

SG内に、S1→S2→・・・→Snなるパスが存在したとする。ここで、Si＝（Ri, Ri＋1）（i＝1, ・・・, n）。このパスをPとする。Rn＋1＝R1である時（サイクルパス）、Si（i＝1, ・・・, n）内の少なくとも1つのセット型のメンバシップクラ

スは、M／Aであってはならない。即ち、$\forall P(Rn+1=R1) \in SG \quad \exists Si \in P$ mem－class（Si）≠M／A。もし、全てが、M／Aであれば、何のレコードオカーランスも、このパス内のレコードオカーランス集合に格納出来なくなってしまう。

D　レコード型の主キー

レコードオカーランス集合R内のレコードオカーランスを、一般にそのデータ項目の値によって識別する事は出来ず、R内での位置又は、Rに関するセットオカーランス内の位置（順番）によって識別できる。従って、リレーショナルモデルにおける主キー（即ち、その値によってR内の全てのレコードオカーランスを一意に識別出来るデータ項目（その集合））概念は、CODASYLモデルには明には存在しない。しかし、次のデータ項目（その集合）は主キーと考えることも出来る。

i）DNA指定を持つCALC項目〔ch．3を参照されたい〕（集合）。このデータ項目は、主インデクスに対応しており、このデータ項目の各値に対応したレコードオカーランスは、1つだけである。このため、これ（CALC　DNA　データ項目と呼ぶ）をレコードオカーランス集合の主キーと考えることが出来る。

ii）Rが、単項セットオカーランス集合Sの子レコードオカーランス集合であり、Sのメンバシップクラスが M／A であり、Rのあるデータ項目（集合）がSについてDNA指定されている時、このデータ項目を（集合）主キーと考え得る。メンバシップクラスがM／Aである単項セットオカーランス集合Sは、Rの全てのレコードオカーランスを、1つのセットオカーランス内に持っている。このため、このセットオカーランス内の全てのレコードオカーランスに対して一意な値を取るデータ項目（即ち、DNAデータ項目）は、全レコードオカーランスにおいても一意な値を取る事になる。従って、このデータ項目は、レコードオカーランス集合の主キーとなる。即ち、

S＝単項セットオカーランス集合（S（SYSTEM，R））

t＝Rのデータ項目（又は、その集合）

$t = pkey(R)$　if　$t = DNA\text{－}item(S) \wedge$

mem－class（S）＝M／A　∧　owner(S)＝SYSTEM

ここで、pkey（R）は、レコードオカーランス集合Rの主キーを表す。

E　リンクレコード型とリンクレコードオカーランス集合

セットオカーランス集合は、異なった2つのレコードオカーランス集合間の1対Nの関係性を表している。これに対して、次の関係性を表すために、リンクレコードオカーランス集合とよばれるレコードオカーランス集合が存在する。

i）同一レコードオカーランス集合間の1対N及びN対Mの関係性

ii）n個（n≧2）のレコードオカーランス集合のn項関係性

R1，・・・，R2を、n個のレコードオカーランス集合とする。これらは、互いに必ずしも異なっている必要はない。これらのn個のレコードオカーランス集合間の関係性（n≧2）

を表すリンクレコードオカーランス集合をLとする。Lの論理的な意味は、次の様である。Lの論理的意味をI（L）として表すと、

$$I(L) \subseteq R1 \times R2 \times \cdots \times Rn \quad (n \geqq 2)$$

CODASYLモデルでは、Lに対応する関係性を持たないので、次の様なセットオカーランス集合を用いて実現される。

R1，・・・，Rn＝n個のレコード型（n≧2）（互いに必ずしも必要はない）

L　　＝レコードオカーランス型

Si　（Ri，L）　（i＝ 1，・・・n）

これは、図4.3の様なデータ構造図によって表れる。ただし、各Siのメンバシップクラスは、M／Aでなければならない。このことは、Lの全レコードオカーランスは、必ず、全ての関係するRi（i＝1，・・・，n）内に、Siについての親レコードオカーランスを持つ事を意味している。

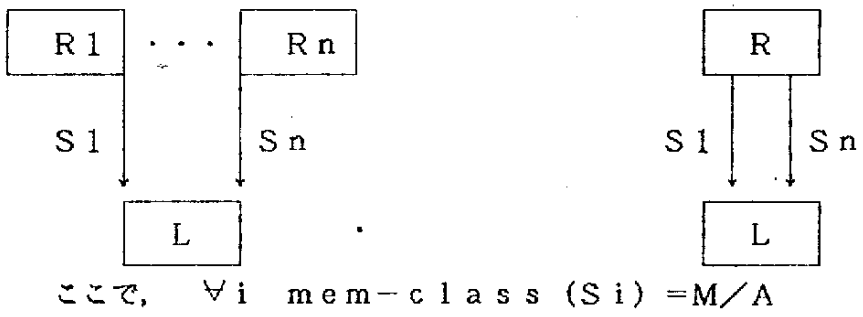


図 4.3　リンクレコード型

4.2.3　ローカル概念スキーマ（LCS）モデル

本節では、ローカル概念スキーマ（LCS）層における各データモデルに対する共通データモデルに必要とされる条件を明らかにする。又、共通データモデルとして、我々が採用したLCSモデルについて、その構造について論じる。

A　共通データモデルの条件

異種データベースシステム（DBS）の同種化とは、各DBSが格納するデータの論理構造を、共通な記述系（即ち、共通データモデル）によって共通表現することである。従って、共通データモデルは、次の条件を満足する必要がある。

1) CODASYLモデル、IMSモデル等の構造を有するデータモデルの論理構造の全てを表すことが出来る。

2) 共通モデルのアクセス言語は、各データモデルのアクセス言語の基本演算を、その基本アクセス演算として備えている。更に、各データモデルの基本アクセス演算によって、そのモデルのインテグリティ条件を保つために生じる更新条件、更新伝播を等しく備えている。

1）は、ＣＯＤＡＳＹＬモデル、ＩＭＳモデルの論理構造が有している事象（例、ＣＯＤＡＳＹＬモデルのレコード型）、関係性（例、ＣＯＤＡＳＹＬモデルのセット型、リンクレコード型）、及び、関係性の有しているインテグリティ条件（例、ＣＯＤＡＳＹＬモデルのメンバシップクラス）という概念を、共通データモデルによって、共通的に表現出来ねばならないことを示している。即ち、共通データモデルは、次の構造を有していなければならない。

i）データモデルの構造定義として、事象を表す構造と、事象間の関係性を表す構造とを、明確に分離して定義出来ねばならない。

ii）関係性の定義として、関係づけられる事象及び関係性のインテグリティ制限が表されねばならない。これは、ＣＯＤＡＳＹＬモデルにおけるセット型のメンバシップクラス概念に対応している。

iii）共通データモデルは、各データモデルの論理構造を共通表現するためのものであり、データ独立性が達成されていなければならない。

2）は、共通データモデルのアクセス言語に対する条件である。ＣＯＤＡＳＹＬモデル、ＩＭＳモデルの基本アクセス演算は、次の様に一般化できる。

i）ある条件を満足する事象の検索。

ii）ある条件を満足する事象の消去と追加。

iii）ある条件を満足する関係性の消去と追加。

iv）ある条件を満足する事象又は、関係性の有する属性の値の置換。

共通データモデルのアクセス言語は、これらの演算を、共通アクセス言語の基本アクセス演算として備えていなければならない。即ち、アクセス言語の原子性は、同一でなければならない。

更に、各データモデルの基本アクセス演算は、そのデータモデルの有するインテグリティ制限を保つために、更新条件を満足する必要があると共に、更新伝播を起す。この更新伝播は、上述したii）とiii）とに対応して、次の様にモデル化出来る。

ii）の更新伝播と更新条件

ii－1）この事象と関係する関係性の消去

ii－2）追加される事象の存在が、ある関係性の存在に依存している時、事象の追加は、同時にこの関係性の追加を必要とさせる。

iii）の更新伝播と更新条件

iii－1）消去される関係性の存在に依存する事象の消去。

iii－2）追加される関係性によって関係づけられる事象は、追加以前に存在していなければならない。

共通データモデルは、上述してきた特徴を満足するものでなければならない。我々は、この共通データモデルとして、ＬＣＳモデルを採用した。この理由は、以下の様である。

1）事象及び関係性の概念を基本としている。

2) 事象間の関係性は、事象と関係性の生存依存関係に関するインテグリティ条件を表せる。

3) 1）及び2）に基づいたアクセス言語として、i）～iv）を満足するものを設定出来る。

以降、ＬＣＳモデルの構造定義（4.2.2）と、アクセス言語（4.2.3）について論じる。

B ＬＣＳモデルの構造定義

まず、ＬＣＳモデルの構造定義について検討する。リレーショナルモデルは、検索に対して、データ独立な非手続的アクセスを提供しているが、リレーション間の更新に対するインテグリティ条件を構造として有していない。ＬＣＳモデルとは、リレーショナルモデルに更新の為のインテグリティ構造を加えたモデルである。このモデルでは、データの論理構造は、基本要素としての事象と、これらの間の関係性とによって表される。ＬＣＳモデルに基づいたデータベースＳは、種々のタイプのリレーションの集合として定義できる。我々は、ＬＣＳモデルのリレーションとしては、次の二種のリレーションを考える。

1) Ｅリレーション　E

2) Ｒリレーション　R

Ｅリレーションは、事象の集合を表し、Ｒリレーションは、関係性の集合を表している。一方、Ｅ－Ｒモデルに基づいたスキーマＳは、上記した2つのリレーションに対応して、次の2つのスキームと、インテグリティ条件ＩＣとから成る。

1) Ｅリレーションスキーム　E

2) Ｒリレーションスキーム　R

(A) Ｅリレーション

Ｅ－リレーションは、事象の集合を表している。ａ０，ａ１，・・・，ａｍを互いに異なった属性とする。各属性ａｉの取り得る値の集合を定義域Ａｉとする。この時、Ｅ－リレーションスキーム E は、属性集合（及び定義域集合）とインテグリティ条件ＩＣＥとからなる。

E （a0〔／A0〕, a1〔／A1〕, ・・・, am〔／Am〕）

with ＩＣＥ

Ｅ－リレーションスキームの外延は、Ｅ－リレーション E と呼ばれ、次の様に定義される。

$$E = \{ e \mid e \in A0 \times A1 \times \cdots \times Am \wedge ICE(e) \}$$

Ｅ－リレーション E の要素ｅは、組又はＥ組と呼ばれる。ｅの属性ａｉの値は、ｅ〔ａｉ〕と表される。

属性集合の中で、属性ａ０は特に主属性（primary attribute）と呼ばれ、次の特徴を有している。

i) 主属性の定義域Ｄ０は、すべてのＥ－リレーションに於いて共通である。

ii) 各Ｅ－リレーション E の主属性は、データベース内の総ての組を一意に識別する。

Ｅ１、Ｅ２をＥ－リレーションとする（互いに、必ずしも異なる必要はない）。

$$\forall e1 \in E1 \quad \forall e2 \in E2 \quad e1〔k1〕 \neq e2〔k2〕$$
$$iff \quad e1 \neq e2$$

従って、主属性は、Eーリレーションの主キーでもある。

各Eーリレーション名Eに対して、以下を定義する。

$$att(E) = Eの属性集合 = \{a0, a1, \cdots, am\}$$
$$patt(E) = Eの主属性集合 = a0$$
$$natt(E) = Eの非主属性集合 = \{a1, \cdots, am\}$$
$$pkey(E) = Eの主キー = patt(E)$$

(B) Rリレーション

n個のEーリレーションスキームを次の様に定義する。

$$\underline{E1} \ (\underline{k1}; A1)$$
$$\underline{En} \ (\underline{kn}; An)$$

ここで、kiはEーリレーションEiの主属性であり、AiはEiの非主属性の属性集合である。Piは主属性piの定義域で、Ajは属性ajの定義域である。E1, ・・・, Enに対するRーリレーションスキームRは、n個の主属性p1, ・・・pnと、m個の非主属性集合a1, ・・・, amと、インテグリティ条件ICRとから成り、次の様に表される。

$$\underline{R} \ (\underline{p1〔/P1〕}, \cdots, \underline{pn〔/Pn〕}, a1〔/A1〕, \cdots, am〔/Am〕)$$

各Rーリレーション名Rに対して、以下を定義する。

$$att(R) = Rの属性集合 = \{p1, \cdots, pn, a1, \cdots, am\}$$
$$patt(R) = Rの主属性集合 = \{p1, \cdots, pm\}$$
$$natt(R) = Rの非主属性集合 = \{a1, \cdots, am\}$$
$$pkey(R) = Rの主キー \subseteq patt(R)$$
$$Es(R) = \{E1, \cdots, En\}$$

Rの主キーは、スキーマの設計時に決定される。

各主属性piは、EーリレーションEiとの関係性を表している。即ち、主属性集合

ｐａｔｔ（Ｒ）とＲに関するＥ－リレーションの集合Ｅｓ（Ｒ）との要素とは、一対一対応している。各主属性ｐｉに対して、以下の関数が定義されている。

ｅ－ｒｅｌ（ｐｉ）＝　Ｅｉ

ｒｏｌｅ　（ｐｉ）＝　ｒｏ$_{i}$

即ち、各主属性ｐｉは、Ｒが関係づけるＥ－リレーションＥｉと共に、Ｒでのロールを表している。ここで、

∀ｐｉ，ｐｊεｐａｔｔ（Ｒ）

ｒｏｌｅ（ｐｉ）≠ｒｏｌｅ（ｐｊ）　ｉｆｆ　ｐｉ≠ｐｊ

Ｒの外延としてのＲ－リレーションＲは、次の様に定義される。

Ｒ＝｛ｒ｜ｒεＰ１×・・・×Ｐｎ×Ａ１×・・・×Ａｍ

∧　ＩＣＲ（ｒ）｝

Ｒ－リレーションＲの要素ｒεＲは、組又はＲ組と呼ばれる。組ｒの属性ａεａｔｔ（Ｒ）の値は、ｒ〔ａ〕と表される。

ｎ個のＥリレーション、Ｅ１，・・・，Ｅｎ（Ｅｉ（ｋｉ，Ａｉ）、ここで、ｋｉ＝ｐｋｅｙ（Ｅｉ）、Ａｉ＝ｎａｔｔ（Ｅｉ）、各Ｅｉは互いに異なっている必要はない）上のＲリレーションＲのインテグリティ条件を考える。又、Ｒリレーションスキームを、次の様に表す。

Ｒ（ｐ１，・・・，ｐｎ，Ａ）

ｐｉは、Ｒの主属性である。Ａは、Ｒの非主属性の集合である。

この時、各主属性ｐｉは、次の条件を満足せねばならない。

1) ∀ｐｉεｐａｔｔ（Ｒ）

ｄｏｍ（ｐｉ）＝ｄｏｍ（ｋｉ）

ここで、

ｋｉ　＝　ｐａｔｔ（Ｅｉ）

Ｅｉ　＝　ｅ－ｒｅｌ（ｐｉ）

2) ∀ｐｉεｐａｔｔ（Ｒ）

Ｒ〔ｐｉ〕⊆Ｅｉ〔ｋｉ〕

即ち、Ｒ内に組ｒが存在する為には、次の条件を満足する組ｅ１，・・・，ｅｎが、各々Ｅ－リレーションＥ１，・・・，Ｅｎ内に存在しなければならない。

ｅ１εＥ１　∧・・・∧　ｅｎεＥｎ　∧　ｒεＲ

ｅ１〔ｋ１〕＝ｒ〔ｐ１〕　∧・・・∧　ｅｎ〔ｋｎ〕＝ｒ〔ｐｎ〕

Ｒ－リレーションＲの意味によって、更新に対して、次の条件と伝播をもたらす。

2－1) 更　新　条　件

組ｒをＲ－リレーションＲに追加する為には、対応する全てのｅ１，・・・．ｅｎが、各々、Ｅ１，・・・，Ｅｎ内に存在せねばならない。関係性は、対応するＥ－リレーシ

ョンの存在無しには、存在し得ない。

2－2）更　新　伝　播

ある組ｅｉ∈Ｅｉが、Ｅ－リレーションＥｉから消去されたならば、同時にｅｉ〔ｋｉ〕＝ｒ〔ｐｉ〕（ｅ－ｒｅｌ（ｐｉ）＝Ｅｉ）なる組ｒが、Ｒ－リレーションＲから消去されねばならない。

3）Ｒ－リレーションＲのある主属性ｐｉは、次の性質を持つものがある。

即ち、全ての時に、　　Ｒ〔ｐｉ〕＝Ｅｉ〔ｋｉ〕

2）に加えて、次の関係が成り立つことを示している。

$$\forall e_i \in E_i \quad \exists r \in R \quad r[p_i] = e_i[k_i]$$

これは、ｅｉとｒとの存在は、互いに依存しあっていることを示している。この性質を有するｐｉをＴタイプと呼び、ａｔｙｐｅ（ｐｉ）＝Ｔと記す。

このことは、次の様な更新の条件と伝播とをもたらす。

3－1）更　新　条　件

ｒをＲに加える為には、ａｔｙｐｅ（ｐｉ）＝Ｔなる全てのｐｉに対して、ｅｉ〔ｋｉ〕＝ｒ〔ｐｉ〕なるｅｉが、同時にＥｉに加えられねばならない。

組ｒをＲ－リレーションＲに追加する為には、対応する全てのｅ１，・・・，ｅｎが、各々、Ｅ１，・・・，Ｅｎ内に存在せねばならない。関係性は、対応するＥ－リレーションの存在無しには、存在し得ない。

3－2）更　新　伝　播

ｒをＲから消去すると、ａｔｙｐｅ（ｐｉ）＝Ｔなる全てのｐｉに対して、ｅｉ〔ｋｉ〕＝ｒ〔ｐｉ〕なる全てのｅｉが、Ｅｉから同時に消去されねばならない。

又、Ｅｉからｅｉが消去されると、ａｔｙｐｅ（ｐｉ）＝Ｔでｅ－ｒｅｌ（ｐｉ）＝Ｅｉなるｐｉを有するＲから、ｒ〔ｐｉ〕＝ｅ〔ｋｉ〕なる全てのｒが、同時に消去されねばならない。

(C)　ＬＣＳ図式

ＬＣＳは、リレーションスキームを表すノードの集合Ｎと、ノード間のリンクの集合Ｌから成るグラフとして表される。ノードには、Ｅ－リレーションを表すＥノードと、Ｒ－リレーションを表すＲノードがある〔図4.4〕

：　Ｅノード

：　Ｒノード

図 4.4　　ノ　ー　ド

各リンク（Ｅｉ，Ｒ）∈Ｌは、ＥｉとＲとの関連を表している。リンクには、次の２種ある。

ⅰ）Ｔリンク

ⅱ）Ｐリンク

Ｔリンクは、次の時、ＥｉとＲ間に存在する。

$$pi \in patt(R)$$
$$e\text{-}rel(pi) = Ei \quad \wedge$$
$$atype(pi) = T$$

Ｐリンクは、次の時、ＥｉとＲ間に存在する。

$$pi \in patt(R)$$
$$e\text{-}rel(pi) = Ei \quad \wedge$$
$$atype(pi) = P$$

これは、図4.5の様に図示される。図中で、$ro_i = role(pi)$である。

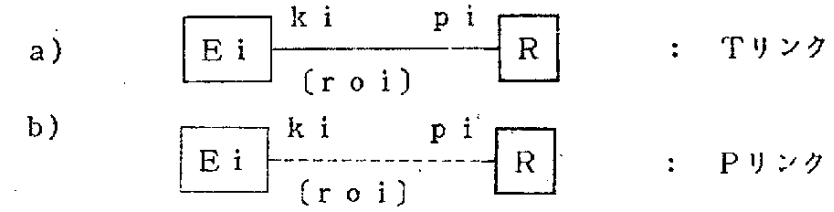


図 4.5　リ　ン　ク

ＬＣＳグラフに対する制限としては、次の点がある。

即ち、ＬＣＳグラフ内に、ループが存在するならば、このループ内のリンクの少なくとも一つは、Ｐリンクでなければならない。

4.2.4　スキーマ変換規則と等価性

Ａ　ＣＯＤＡＳＹＬ－ＬＣＳ変換

本項では、ＣＯＤＡＳＹＬスキーマからＬＣＳへの変則規則（Ｈ）について考える。ＣＯＤＡＳＹＬスキーマとの対応を考える時に、ＬＣＳモデルに於ける主キーの対応が重要になる。ＬＣＳモデルの主キーは、ＬＣＳリレーション内の組を、一意に識別する属性又は属性集合である。又、各リレーションは、必ず主キーを有している。これに対して、ＣＯＤＡＳＹＬスキーマでは、レコードオカーランス集合内の各レコードオカーランスを一意に識別できるデータ項目又はその集合は、一般に存在しない。4.2.2では、レコードオカーランス集合の主キーとなり得る幾つかのデータ項目について示した。しかし、こうしたデータ項目は、一般には、存在し得ない。この為、我々は、レコードオカーランス集合内の個々のレコードオカーランスを一意に識別する機構として、データベースキー（ｄｂ－キー）を用いることにする。ｄｂ－キ

ーは、１つの実行単位内で、個々のレコードオカーランスを一意に識別する値を取る。しかし、ユーザは、この値を実行単位内で参照する事は出来るが、出力して視ることは出来ない。この概念は、ＬＣＳモデルに於ける主キーと同じである。

1) Ｅーリレーションは、ただ１つの主キー属性を有する。

2) リレーションの主キーの値をユーザは、視ることは出来ない。

従って、ｄｂーキーを、ＬＣＳにおける主キーと対応させる事が出来る。

以下にスキーマ変換規則を示す。

＊＊ ＣＯＤＡＳＹＬ － ＬＣＳ スキーマ変換規則（Ｈ） ＊＊

1. レコード型 Ｒ （ｔｉ，・・・，ｔｎ） に対して

Ｅーリレーションスキーム

Ｅ（Ｒ） （@Ｅ(Ｒ)／ｄ（Ｒ），ａｔｔ（ｔ１），・・・，ａｔｔ（ｔｎ））

を生成する。

ここで、

Ｅ（Ｒ） ＝レコードオカーランス集合Ｒに対応したＥーリレーション名

ａｔｔ（ｔｉ）＝データ項目ｔｉに対応した属性名

@Ｅ（Ｒ） ＝ｄｂーキーの値を仮想的に取る仮想的な主属性属性

＝ｐａｔｔ（Ｅ（Ｒ））

ｄ（Ｒ） ＝主属性@Ｅ（Ｒ）に対応した定義域

＝レコードオカーランス集合Ｒ内のレコードオカーランスのｄｂーキーの値の集合

2. セット型 Ｓ（Ｒ１，Ｒ２）（（但しＲ２は、リンクレコード型ではない）に対して、Ｒーリレーションスキーム

Ｒ（Ｓ） （@Ｅ(Ｒ１)／ｄ１，@Ｅ(Ｒ２)／ｄ２

ここで、

$$\left.\begin{array}{l} di = dom(pkey(E(Ri))) \\ @E(Ri) - patt(E(Ri)) \end{array}\right\} \quad (i=1,\ 2)$$

以下のセットがなされる。

ｐｋｅｙ（Ｒ（Ｓ））← @Ｅ(Ｒ２)；

ｒｏｌｅ（ｐ１） ← Ｏ；

ｒｏｌｅ（ｐ２） ← Ｍ；

ｅーｒｅｌ（ｐｉ） ← Ｅ（Ｒｉ） （ｉ＝１，２）；

$$atype(p1) \leftarrow P;$$

$$atype(p2) \leftarrow \begin{cases} T & if \quad mem-class(S)=M/A \\ P & otherwise \end{cases}$$

3. リンクレコード型 L (t1, ・・・, tm) と、

セット型 Si(Ri, L) (i=1, ・・・, n) に対して、

Rーリレーションスキーム

R(L) (p1/d1, ・・・, pn/dn, att(t1), ・・・, att(tm))

が生成される。

ここで、

$$\left.\begin{array}{l} di = dom(patt(E(Ri))) \\ pi = pattn(Si, Ri, R) \\ R = \{Ri \mid i=1, \cdots, n\} \end{array}\right\} \quad (i=1, \cdots, n)$$

この時、以下のセットがなされる。

$$\left.\begin{array}{l} role(pi) \leftarrow Si; \\ e\text{-}rel(pi) \leftarrow Ei; \\ atype(pi) \leftarrow T; \end{array}\right\} \quad (i=1, \cdots n)$$

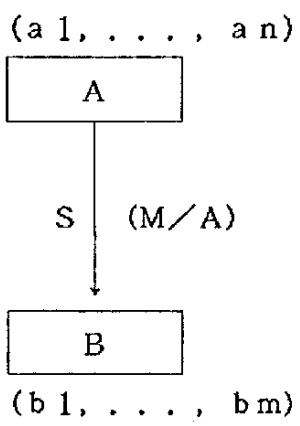
となる。

pattn(Si, Ri, R)は、属性名をつくる関数である。

$$pattn(Si, Ri, R) = \begin{cases} pkey(E(Ri)) & if \quad R内にRiと同じレコード型は無い。 \\ Si.patt(E(Ri)) & (Siとpatt(E(Ri))との連結) \\ otherwise & \end{cases}$$

図4.6は、CODASYLスキーマからLCSへの変換規則をまとめてある。

CODASYLスキーマ



LCS

E(A) (@E(A), att(a1), ・・・, att(an))

E(B) (@E(B), att(b1), ・・・, att(bm))

R(S) (@E(A), @E(B))

with role(@E(A))=O

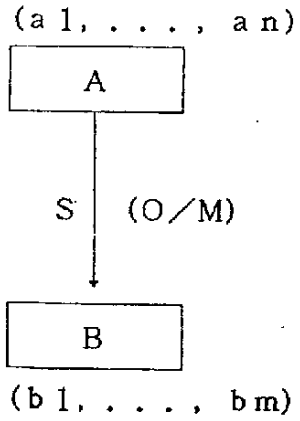
role(@E(B))=M

e−rel(@E(A))=E(A)

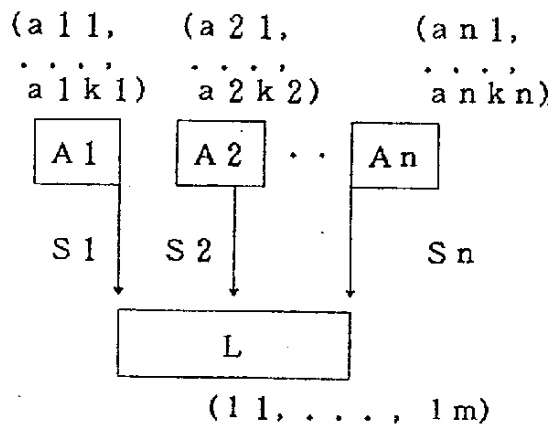
e−rel(@E(B))=E(B)

atype(@E(A))=T

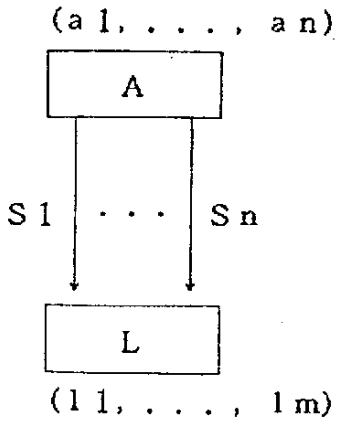
atype(@E(B))=P



R(S) (@E(A), @E(B))
with atype (@E(A))=P
atype (@E(B))=P



R(L) (@E(Ai), ⋯, @E(An),
att(1i), ⋯, att(lm))
with atype(@E(Ai))=P(i=1,
⋯, n)
role(@E(Ai)=Si



R(L) (S1 @E(A), ⋯, Sn@E(A),
att(l1), ⋯, att(lm))
with role(Si@E(Ai))=Si

図 4. 6　変　換　規　則

B　ＬＣＳ　－　ＣＯＤＡＳＹＬスキーマ変換

次に、ＬＣＳから、ＣＯＤＡＳＹＬスキーマへの変換規則（Ｉ）について考える。

ＬＣＳは、Ａの条件を満足しているとする。Ｉは、変換規則Ｈの逆関数と考えられる。

変換規則Ｉ

1. Ｅリレーションスキーム Ｅ （ｐ, ａ１, ・・・, ａｍ）に対して,

レコード型　Ｒ（Ｅ）（ｉｔｅｍ（ａ１）, ・・・, ｉｔｅｍ（ａｍ））を生成する。

ここで、Ｒ（Ｅ）＝Ｅに対応したレコード型名

ｉｔｅｍ（ａｉ）＝ａｉに対応したデータ項目

2. Ｒリレーションスキーム Ｒ （ｐ１, ｐ２）

ｗｉｔｈ　ｅ－ｒｅｌ（ｐｉ）＝Ｅｉ（ｉ=1,2）

ｒｏｌｅ（ｐ）＝Ｏ　ｒｏｌｅ（ｐ２）＝Ｍ

に対して、セット型 Ｓ（Ｒ）（Ｒ（Ｅ１）, Ｒ（Ｅ２））

ここで、Ｓ（Ｒ）（Ｒ（Ｅ１）, Ｒ（Ｅ２））

この時、$\mathrm{mem-class}(S(R)) = \begin{cases} M/A & \text{if type}(P2)=T \\ O/M & \text{if type}(P2)=\phi \end{cases}$

Ｅ１（@Ｅ１, ａ１１, ・・・, ａ１ｍ１）

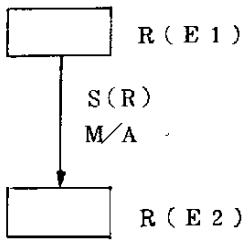
Ｅ２（@Ｅ２, ａ２１, ・・・, ａ２ｍ２）

Ｒ（ｐ１, ｐ２）

ｗｉｔｈ　ｒｏｌｅ（ｐ１）＝Ｏ　atype（Ｐ２）=Ｔ

ｒｏｌｅ（ｐ２）＝Ｍ

ｅ－ｒｅｌ（ｐｉ）＝Ｅｉ



3. Ｒリレーションスキーム Ｒ （ｐ１, ・・・, ｐｎ, ａ１, ・・・, ａｍ）

ｗｉｔｈ　ｅ－ｒｅｌ（ｐｉ）＝Ｅｉ

ｒｏｌｅ（ｐｉ）＝Ｓｉ　の時

セット型の集合　Ｓ（Ｓｉ）（Ｒ（Ｅｉ）, Ｒ（Ｒ））（ｉ＝１, ・・・, ｎ）と

リンクレコード型　Ｒ（Ｒ）（ｉｔｅｍ（ａ）, ・・・, ｉｔｅｍ（ａｍ））とを生成する。

この時、ｍｅｍ－ｃｌａｓｓ（Ｓ（Ｓｉ））＝Ｍ／Ａである。

Ｅｉ（@Ｅｉ, ａｉ１, ・・・, ａｉｍｉ）（ｉ＝１, ・・・, ｎ）

Ｒ （ｐ１, ・・・, ｐｎ, ａ１, ・・・, ａｍ）

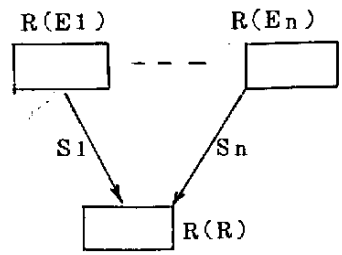
ｗｉｔｈ

ｅ－ｒｅｌ（ｐｉ）＝Ｅｉ

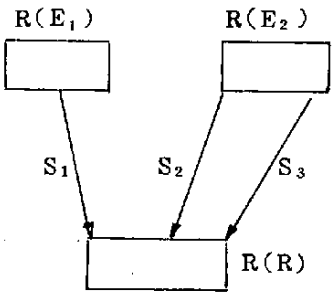
ｒｏｌｅ （ｐｉ）＝Ｓｉ

ｔｙｐｅ （Ｒ）＝Ｇ

ここで、　ｍｅｍ－ｃｌａｓｓ（Ｓｉ）＝Ｍ／Ａ



E 1 (@E 1, a 1)
E 2 (@E 2, a 2 1, a 2 2)
R (p 1, p 2, p 3, a)
w i t h
e－r e l (p 1)＝E 1　r o l e (p 1)＝S 1
e－r e l (p 2)＝E 2　r o l e (p 2)＝S 2
e－r e l (p 3)＝E 2　r o l e (p 3)＝S 3



H及びIは、E－Rスキーマの要素とCODASYLスキーマの要素とを、1対1対応させていることから、Iは、Hの逆関数（H^{-1}）である。

即ち、CODASYLスキーマSに対して、H（S）＝Cとする。

この時、　I（C）＝Sである。

C　等　価　性

次に、変換規則Hによって、CODASYLスキーマCから生成されたローカル概念スキーマ（LCS）Sが、Cと等価であることを示す。4.2.1で述べた様に、SとCとが等価であるためには、情報と更新伝播の保存性を示さねばならない。

a　情報保存性

CとSとが情報保存しているためには、各々の外延CとSとの要素が、1対1対応していることである。

1) レコード型 R（t 1, ・・・, t n）から生成されたE－リレーションスキームを、
E (@E, a 1, ・・・, a n) とする。

ここで、E（R）＝E, a t t (t i)＝a i, p（R）＝@Eであるとする。

この時、Rの個々のレコードオカーランスと、E内の個々の組とは、1対1対応している。即ち、

$\forall r \in R \quad \exists ! e \in E$

$\forall e \in E \quad \exists ! r \in R$

d b－k e y (r)＝e〔@E〕 ∧ r〔t 1〕＝e〔a 1〕 ∧・・・∧
r〔t n〕＝e〔a n〕

2) セット型 S（R 1, R 2）とRリレーションスキーム R（@E 1, @E 2）

ここで、E（R 1）＝E 1,　E（R 2）＝E 2とする。

この時、

$\forall (r1, r2) \in S \quad \exists ! r \in R$

$\forall r \in R \quad \exists ! (r1, r2) \in S$

e 1〔@E 1〕＝r〔@E 1〕 ∧ e 2〔@E 2〕＝r〔@E 2〕∧ r ∈ R∧
d b－k e y (r 1)＝e 1〔@E 1〕 ∧ d b－k e y (r 2)＝e2〔@E2〕

であるので、SとRの要素は、1対1対応している。

3) セット型 S1(R1, L), ・・・, Sn(Rn, L)

とリンクレコード型 L(t1, ・・・, tm)

とRリレーションスキーム R(p1, ・・・, pn, a1, ・・・, am)

ここで、role(pi)=Si, E(Ri)=Ei, att(ti)=aiとする。

この時、

$\forall(r1, 1)\in S1 \quad \cdots \quad \forall(rn, 1)\in Sn$

$\exists!\, r\in R \quad \exists!\, e1\in E1 \cdots \quad \exists!\, en\in En$

$\forall r\in R \quad \exists!(r1, 1)\in S1 \quad \cdots \quad \exists!(rn, 1)\in Sn$

$r[a1]=1[t1] \wedge \cdots \wedge r[am]=1[tm] \wedge$

db-key(r1)=e1[@E1] ∧・・・∧

db-key(rn)=en[@En]

即ち、レコードオカーランスri∈Ri(i=1, ・・・, n)の間の1つの関係性は、R内の一つの組に、1対1対応している。

従って、CODASYLスキーマCと、LCS Sとは、互いに、その外延の要素が1対1対応しているので情報保存されていることがわかる。

b 更新保存性

CODASYLモデルとLCSモデルにおける更新演算の等価性について示す。各モデルにおける基本更新演算(i.e.ユーザにとって、atomicな演算単位)は、次の様にモデル化できる。

i) 更新対象を更新される為に規定される必要があるデータベース要素の集合(更新条件)

ii) 更新対象の更新

iii) この要素の更新によって生じる他の要素への更新(更新伝播)

以下、各々の基本演算について調べる。

1) 消 去

1−1) Eリレーション E の組 e の消去

LCSデータベースからeを消去すると、eと関連する全てのR−リレーションRの組r(r[p]=e[k]∧e−rel(p)=e)の消去をもたらす。一方、rの消去は、タイプTの主属性qと関連する全てのE−リレーションEの組(r[q]=e[k]∧e−rel(q)=e)の消去をもたらす。eの消去によって、消去される要素の集合PS(e)は、次の様に定義される。

$$PS(e) = \{\, r,\ PS(r) \mid r\in R \wedge p\in patt(R) \wedge r[p]=e[k] \wedge e\text{-}rel(p)=E \,\}$$

$$PS(r) = \{\, e',\ PS(e') \mid e'\in E' \wedge r[p]=e[k] \wedge e\text{-}rel(p)=E' \wedge atype(p)=T \wedge p\in patt(R) \,\}$$

一方、eに対するCODASYLデータベース内のレコードオカーランスをr∈Rとする。rの消去は、まず、rと関連する全てのセットオカーランスからrを消去する事をもたらす。もし、rがメンバシップクラスM／Aのセットオカーランス集合Sの親レコードオカーランスであるならば、rのSに関する全ての子レコードオカーランスの消去をもたらす。ここで、PC(r)を、rの消去によって、消去の及ぶ要素の集合とする。

$$PC(r)=\{s, PC(s) \mid s\in S \wedge (R=member(S) \vee R=owner(S))\}$$

$$PC(s)=\{r', PC(r') \mid R=owner(S) \wedge s=(r, r') \wedge mem\text{-}class(S)=M/A\}$$

1－2) R－リレーションRの組rの消去

1－1）と同様である。

2) 追　加

1－1)　EリレーションEに、組e追加する。

EリレーションEに組e追加する為には、EとTリンクで結合されたR－リレーション R1，・・・，Rnにも、同時に、eとの関係性が追加されねばならない。

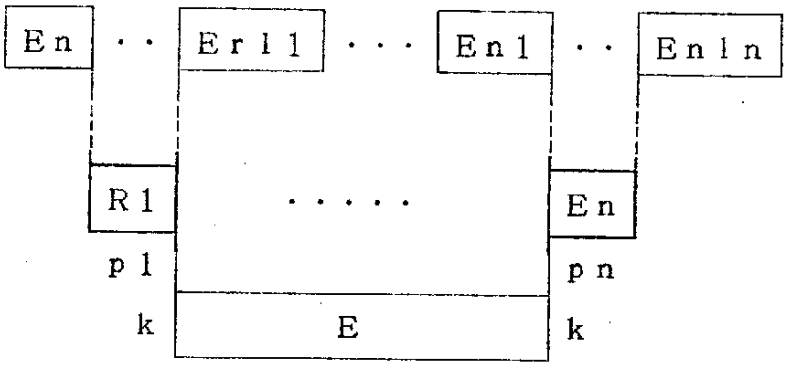


図 4. 8

この為には、R1，・・・，Rnの関係するE－リレーションE11，・・・，Enln内の対応する各々の組e11，・・・，enlnを識別する必要がある。この集合をIS(e)とする。

$$IS(e) = \{ e11, \cdots, e111, \cdots, enln \}$$

又、eの追加と共に、以下のレコードオカーランスの追加がなされねばならない。

$$PS(e) = \{ (e11, \cdots, e111, e)\cdots (en1, \cdots, enln, e) \}$$

一方、レコードオカーランス集合Rに、レコードオカーランスrを追加する為には、メ

ンバシップクラスがM／Aであるセットオカーランス集合Sのセットオカーランスへ同時に追加されねばならない。

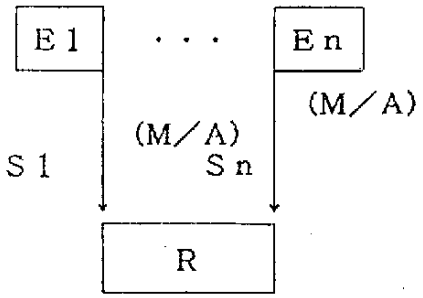


図 4. 9

図4.9において、E1，・・・，Enのレコードオカーランスの識別がなされねばならない。これの集合をIC(r)とする。

$$IC(r) = \{e1, \cdots, en\}$$

更に、rの追加と共に次の追加がなされなければならない。

$$PC(r) = \{(e1, r), \cdots, (en, r)\}$$

2－1) Rーリレーション Rへの組rの追加

Rリレションへの追加の条件としては、Rが主キーで且つ、Tタイプの主属性を持つ時、これへの追加を許さないことがある。これは、CODASYLモデルにおける、M／Aクラスのセット型に、新たな追加を許さないことに対応している。Rに組rを追加するためには、Rの関連する全てのEリレーション

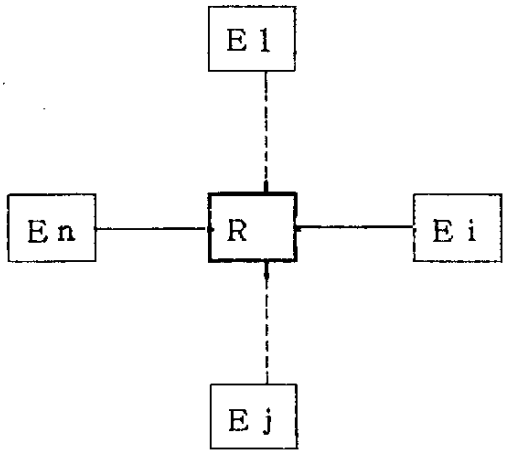


図 4. 1 0

の組を識別せねばならない。この集合をISC(r)とする。

$$IS(r) = \{e \mid e \in E \wedge patt(R) \wedge e\text{-}rel(p) = E\}$$

(e1, ・・・, en)をRに追加することになる。

CODASYLモデルは、セットオカーランス、又は、リンクレコードオカーランスの追加に対応している。

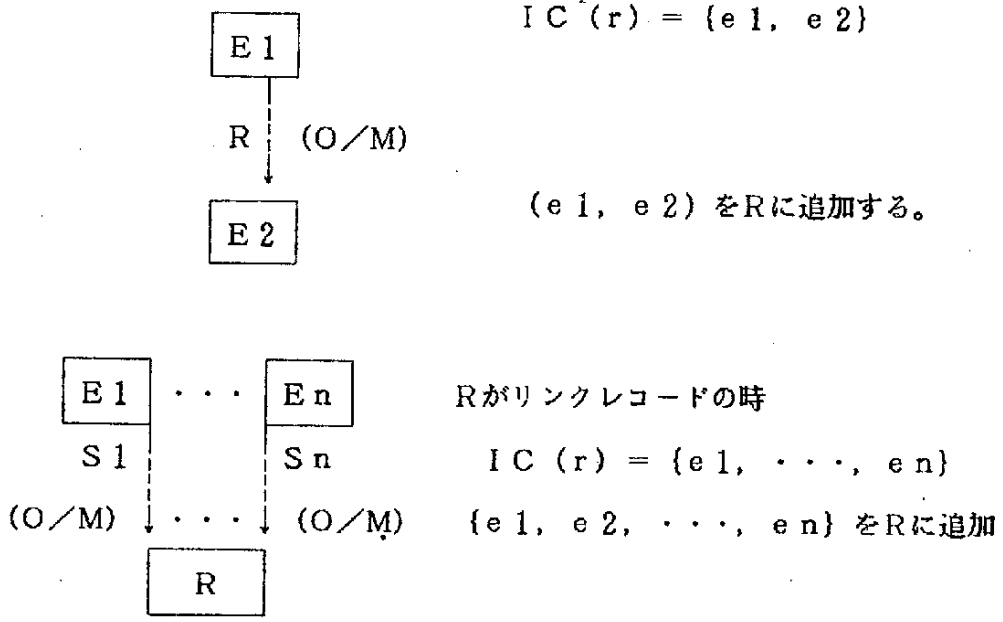


図 4.11

3) 置　換

3－1) Eリレーションの組eをe'で置換

主属性値の変換を許さない(e(k)=e(k))なので、単に属性の値の変換であり、これは、対応するレコードオカーランスrの値の変換に対応

3－2) Rリレーションの組rとr'で置換

LCSモデルでは、単一主キー又は、Tタイプの主属性値の変換を許さない。このことは、CODASYLモデルでは、M／Aのセット型の子オカーランスを置き換えることを禁止していることである。即ち、M／Aの時、あるr"に対して、子オカーランスrをr'にすることは、 rが消去されるとともに、r'を生成することを意味してしまう。我々は、LCSにおける基本更新演算は、同種のCODASYL基本演算に変換されるとの前提を設けているからである。

4.2.5　例

ＰＲＤＢＳ（図4.12）に対して、図4.13の様なＬＣＳが生成される。

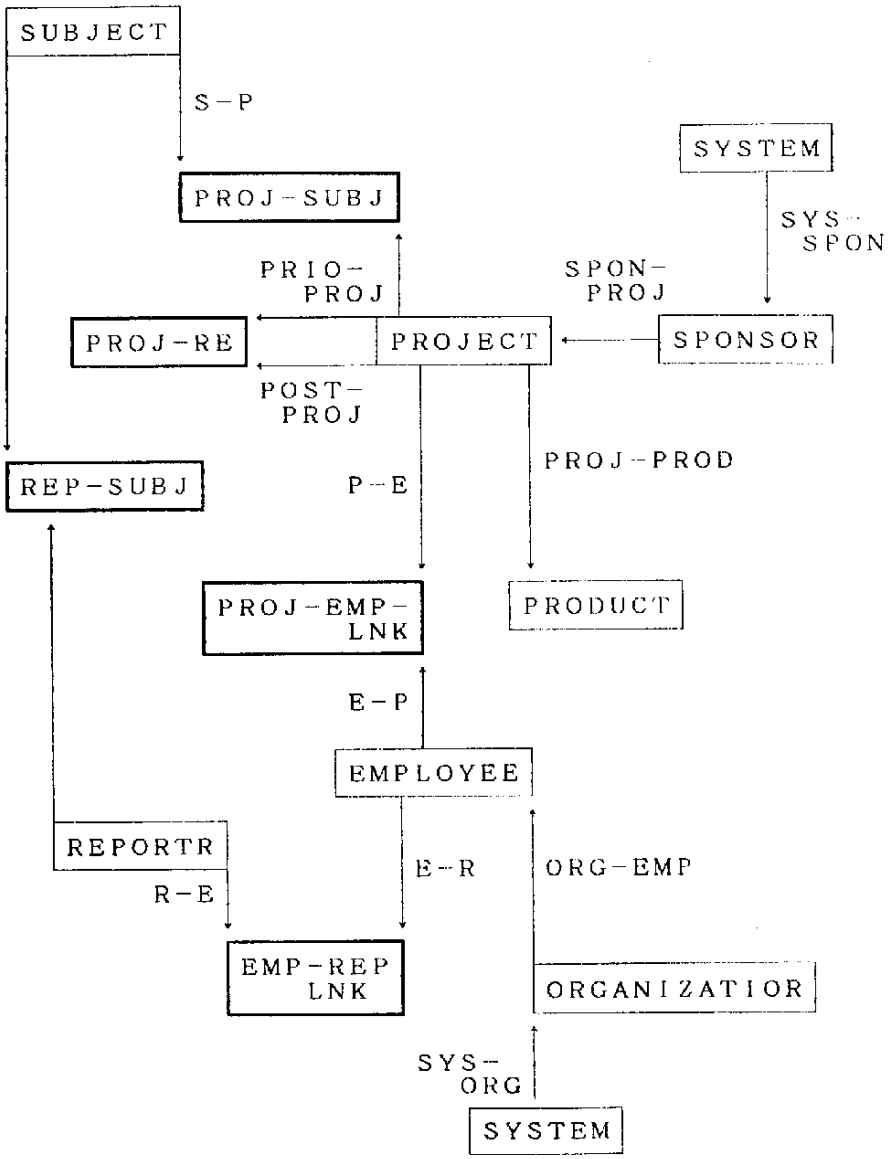


図4.12　ＰＲＤＢＳのスキーマ

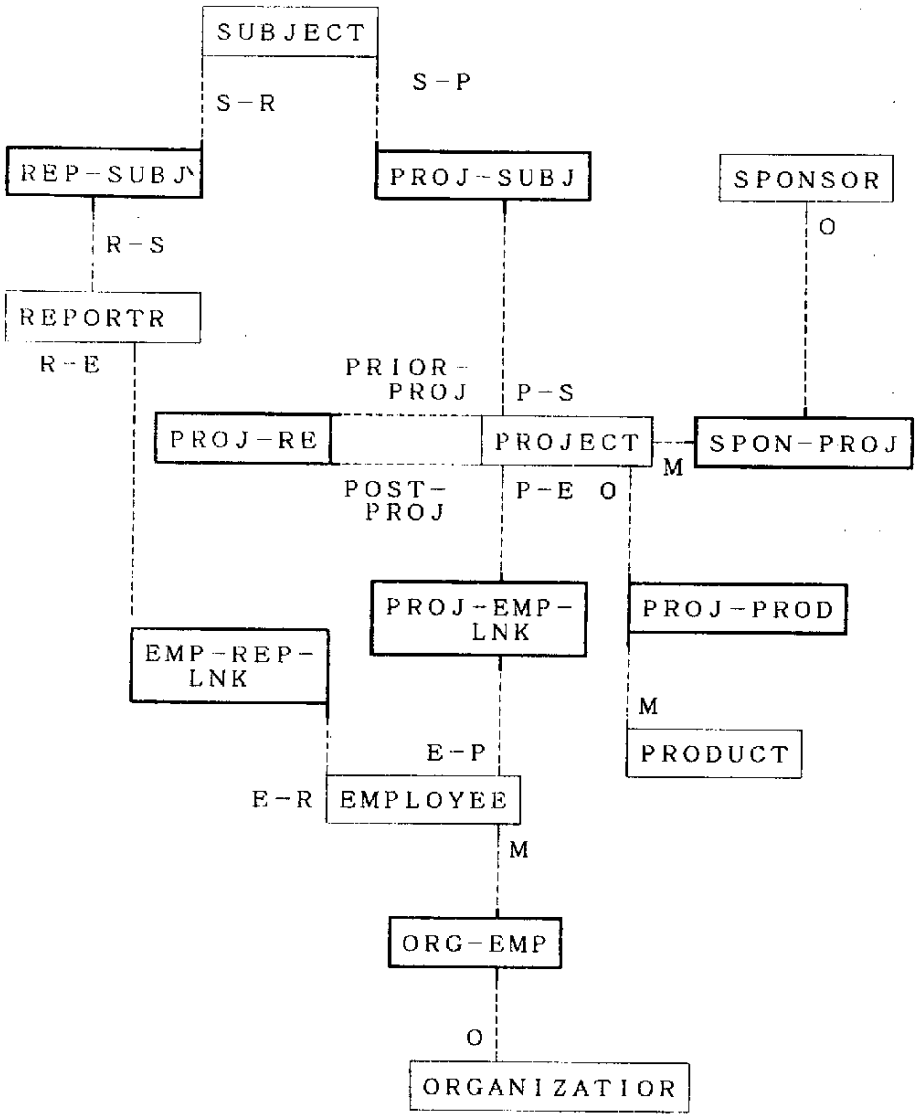


図4.13　図4.12のLCS

4.2.6 ま と め

本節では、異種データベースシステム（ＤＢＳ）から成る分散型データベースシステム（ＤＤＢＳ）を実現するための第１段階として、ＤＢＳの異種データモデルに基づいたスキーマ（ローカル内部スキーマ又は、ＬＩＳ）に対して、ＤＤＢＳ全体に対して共通のモデルに基づいたスキーマ（ローカル概念スキーマ又は、ＬＣＳ）を生成する同種化について論じた。このなかで、次のことを行った。

i) ＣＯＤＡＳＹＬモデルを、論理構造（概念スキーマレベル）と物理構造（内部スキーマレベル）とに分離した。更に、各々に対して、形式的な定義を行った。又アクセス言語の構文と意味を形式化した。

ii) ＬＣＳモデルを、ＣＯＤＡＳＹＬモデル、ＩＭＳモデル等の構造を有したデータモデルの論理構造の共通表現のためのモデルとして設定し、その構造とアクセス言語とを形式的に定義した。アクセス言語では、特に、構造の有するインテグリティ条件を保つために生じる副作用を明らかにした。

iii) スキーマ変換（同種化）の目標を設定し、ＣＯＤＡＳＹＬスキーマから、ＬＣＳモデルスキーマへの変換アルゴリズムを示し、目標を達成していることを示した。

4.3 更新演算変換

ここでは、ＬＣＳに基づいた非手続的ＱＵＥＬ更新文からＣＯＢＯＬ　ＤＭＬプログラムの生成について論じる。

4.3.1 CODASYL DML

本項ではＣＯＤＡＳＹＬモデルにおけるデータ操作（更新）機能についてのみ述べる、検索については 3.2.1 を参照されたい。

更新演算としては、レコードオカーランス集合、セットオカーランス集合に対する追加、消去、置換がある。まず、ここでは、次の様なシステム演算を定義する。ついで、ＣＯＤＡＳＹＬモデルのアクセス演算を、これらによって表すことにする。Ｓ、Ｒを各々レードオカーランス集合、セットオカーランス集合とする。又、次の記法を定義する。

set−m(R) = { S | member(S) = R }

set−o(R) = { S | owner (S) = R }

A 基本システム演算

基本システム演算は、レコードオカーランス集合Ｒとセットオカーランス集合Ｓとに対して、現在子の指すレコードオカーランスを更新する。ここで、Ｒ1＝ｏｗｎｅｒ（Ｓ）、Ｒ2＝ｍｅｍｂｅｒ（Ｓ）とする。

i) 消　去

a) ｄｅｌＲ（ｒ，Ｒ）

Ｒから、レコードオカーランスｒを消去する。

意味　R ← R − { r } ;

b) delS(s, S);

Sから、セットオカーランスsを消去する。

意味　S ← S − { s } ;

ii) 追　加

a) appR(r, R)

Rに、レコードオカーランスrを格納する。

意味　R ← R ∪ { r } ;

b) appS(s, S) ;

Sに、セットオカーランスsをSに追加する。

意味　S ← S ∪ { s } ;

iii) 置　換

a) repR(r', r, R) ;

Rのレコードオカーランスrをr'によって置換する。

意味　R ←(R − { r }) ∪ {r'} ;

b) repS(s', s, S) ;

Sのセットオカーランスsをs'によって置換する。

意味　S ←(S − { s }) ∪ {s'} ;

IV) 更新演算の原子性の保証

sbegin

S1;・・・;Sn

send;

ここで、S1, ・・・, S2は、基本システム演算である。この時、Siはこの順序で実行されると共に、他のシステム演算の実行は許されない。

B 更 新 演 算

CODASYLモデルのアクセス言語（DML）による更新は、セットオカーランス集合のメンバシップクラスの有するインテグリティ条件を満足する様に行われる様に行われる必要がある。これは、あるレコードオカーランス、又はセットオカーランスの更新が、インテグリティ条件を満足する様に、他のレコードオカーランスの更新を行うという更新伝播（update propagation）と更新を行う為の条件（更新条件）によって表される。以降、レコードオカーランス集合とセットオカーランス集合に対する更新演算と、その更新伝播と条件とについて検討する。更新演算として、親言語に依存しないアクセス言語をここで定義する。この前に次の記法を定義する。

MA−m(R)= { S | member(S)=R ∧
mem−class(S)=M/A }

MA－o(R)＝ { S | owner (S)=R ∧
mem－class(S)=M/A }

R1 ＝ owner(S)

R2 ＝ member(S)

j) 消　去

a) delete　R;

Rから、現在子の指すレコードオカーランス(r＝current(R))を消去する。

更新条件　無

更新伝播

・Rの関連するセットオカーランス集合から、current(R)を除く。

・RがM/AのSの親レコードオカーランス集合の時、対応するセットオカーランスと共に、curent(R)のSについての子レコードオカーランスを消去する。

sbegin

r ← find R;

Rdelete(r, R);　current ← ⊥

send;

procedure　Rdelete(r, R);

sbegin

for ∀S∈set－m(R) ∪ set－o(R)　do

for ∀s∈S (r＝member(s) ∨ r＝owner(s)) do

Sdelete(s, S);

delR(r, R)

send of Rdelete;

COBOL DML

・erase　R　permanent.

b) delete　S;

Sから、current(S)を消去する。

更新条件　無

更新伝播

・SがM/Aの時、対応するセットオカーランスと共に、current(R)のSについての子レコードオカーランスを消去する。

sbegin

s ← find ・・・ within　S;

Sdelete(s, S);　current(S) ← ⊥

send;

```
procedure   Sdelete(s, S);
sbegin
  if    mem−class(S)=M/A,
    then   Rdelete(member(s), member(S));
         Sdelete(s, S);
  delS(s, S)
send of Sdelete;
```

COBOL DML

・disconnect R2 from S.　(mem−class(S)≠M/A)

・erase R2 permanent.　(mem−class(S)=M/A)

ii) 追　加

a) append r to R;

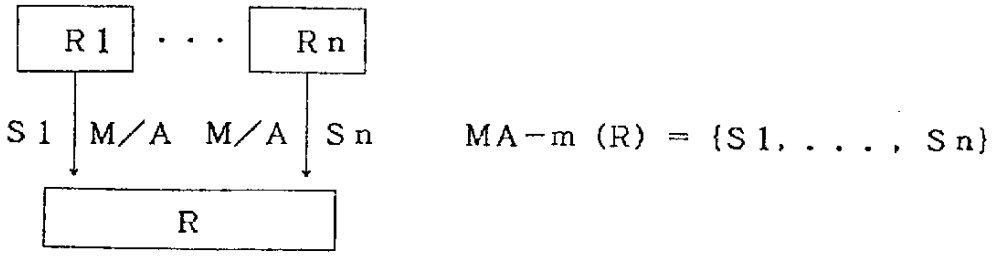
レコードオカーランスrを Rに格納する。

更新条件

・RがメンバシップクラスM／AのSの子レコードオカーランス集合の時 Sのあるセットオカーランスに同時に、このレコードオカーランスは追加されねばならない。

更新伝播　無

```
sbegin
    r1 ← find R1
    . . .
    rn ← find Rn
    appR(r, R);
    appS((r1, r), S);
    . . .
    appS((rn, r), S);
send;
```



COBOL DML

・store R.　if MA−set−m(R)≠φ

・find owner(S). store R. otherwise

b) append S.

Sにr1＝current(R1)とr2＝current(R2)を設定する。

更新条件

・r1、r2が各々R1、R2に存在せねばならない。

更新伝播 無

abegin if mem−class(S)≠M/A ∧
$f S^{-1}$(current(R2))＝φ,
then appS(s, S)

aend;

COBOL DML

・connect R2 to S.

iii) 置 換

a) replace R by r;

Rのcurrent(R)を、レコードオカーランスrで置換する。

更新条件 無

更新伝播 無

sbegin
r' ← find R;
repR(r, r', R)
send;

COBOL DML

・modify R.

b) replace S;

r2＝current(R2)を、Sの現在属するセットオカーランスから除き、r1＝current(R1)を親レコードオカーランスとするセットオカーランスに追加する。

更新条件

・r1は、R2に存在せねばならない

更新伝播 無

sbegin i←current(owner(S));
current(owner(S))←$f S^{-1}$(current(member(S)));
del S; current(owner(S))←i; app S
send;

COBOL DML

・disconnect R2 from S.

connect R2 to S. if mem−class(S)≠M／A

・modify R2 only S membership. otherwise

4.3.2 ＬＣＳモデルのアクセス言語

本項では、ＬＣＳモデルのアクセス言語について考える。まず、アクセス言語の検索機能について論じ、ついで更新機能について論じる。

A 検　索

ＬＣＳモデルにおける検索とは、リレーショナルモデルと同じく、ある条件を満たす組の集合を求めることである。リレーショナルモデルにおける問合せ言語ＱＵＥＬを用いて問合せを記述できる。

range　　(x1, X1)・・・(xm, Xm);

q: retrieve into R(ral=aexp$_1$, ・・・, rak=aexp$_k$)

where　qual;

var(q)={xi | i=1・・・, m}

ti∈var(q)

X1, X2, ・・・, Xmは、E−、又はR−リレーションのどれかである。

qualは、ＱＵＥＬと同様に定義される。

問合せqの意味次の様である。

$$R=\{\ e\ \mid\ x1\in X1\ \wedge\cdots\wedge\ xm\in Xm\ \wedge e=(aexp_1,\ \cdots,\ aexp_k)\ \wedge\ qual(x1,\ \cdots,\ xm)\ \}$$

リレーショナルモデル問合せに対して、ＬＣＳ問合せは次の制限を持っている。

1) 主属性の値は、問合せ内で参照する事は出来るが、結果として出力する事は出来ない。

2) 主属性間の結合において、＝と≠の比較演算のみが許される。

B 更　新

次に、ＬＣＳモデルにおける更新演算について考える。更新では、ある事象組、関係性組に対する更新に加えて、Ｒリレーションの保有するインテグリティ制限を守るための更新条件と、これによって生じる更新伝播についても検討せねばならない。更新演算を論じる前に、以下の基本システム演算を定義する。

1) 基本システム演算

i) 消　去

a) 事象組eの消去

構文　　delE (e, E);

意味　　E ← E− {e} ;

b) 関係性組rの消去

delR (r, R) ;

R ← R− {r} ;

ii) 追　加

a) 事象組eの追加

構文　　appE (e, E)

意味　　E ← E ∪{e} ;

b) 関係性組rの追加

構文　　appR (r, E)

意味　　R ← R ∪{r} ;

iii) 置　換

a) 事象組eを、e'と置換する。

構文　　repE (e', e, E)

意味　　E ← (E−{e}) ∪{e'} ;

b) 関係性組rを、r'と置換する。

構文　　repR(r', r, R)

意味　　R ← (R− {r})∪{r'}

iv) 原　子　性

構文　　sgebin

op1;・・・; opn

send;

opiは、基本演算である。

意味　　op1, ・・・, opnは、この順に原子的に実行される。

2) 基本更新演算

LCSモデルにおける更新演算の形式は、リレーショナルモデルにおけるQUELとほとんど同じである(append文が異なっている)。リレーショナルモデルの更新が、1つのリレーションの組に対するread/writeを行うものであるのに対して、LCSモデルでは、これに加えて、この更新を行うための条件(更新条件)と、この更新によって生じる更新伝播とを明確にする必要がある。

ここで、E−リレーションE、R−リレーションRに対して、以下の記法を定義する。

Rs(E) = {R | pεpatt(R) e−rel(p)=E }

RTs(E)= {RR| pεpatt(R) ∧ atype(p)=T
∧ e−rel(p)=E }

Es(R) = {E | pεpatt(R) ∧ e−rel(p)=E }

ETs(R)= {E | pεpatt(R) ∧ e−rel(p)=E
∧ atype(p)=T }

$$ERTs(E,R)=\begin{cases} Es(R)-\{E\} & if \quad R \varepsilon RTs(E) \\ \phi & otherwise \end{cases}$$

2－1）消　去

a）事象組の消去

構文　　range　（e, E）;
　　range　（x1, X1）（x2, X2）・・・(xm, Xm);
　　delete　e　where　qual;

意味　Eリレーション E から、条件式 qual を満足する事象組 e の集合を消去する。

$$E \leftarrow E - \{ e \mid e \varepsilon Ei \wedge qual(e, x1, \cdots, xm) \wedge x1 \varepsilon X1 \wedge \cdots \wedge xm \varepsilon Xm \};$$

更新伝播

i）消去される事象組 e と、他の事象組とを関係づける全ての R －リレーション R の全ての組 r を消去する。

```
sbegin
  T ← { e | e∈E ∧ qual(e) } ;
  for ∀e∈E do
                Edelete(e, E)
send;
procedure    Edelete(e, E);
sbegin
  for ∀R∈Rs(E) do
    for ∀r∈R( r〔p〕 = e〔k〕 ∧ e－rel(p)=E )
      do
          Rdelete(r, R);
    delE(e, E)
send of Edelete;
```

Rdelete（r, R）は、次のb）で定義される。

b）関係性組の消去

構文　range　（r, R）;
　　range　（x1, X1）（x2, X2）・・・（xm, X, ）;
　　delete　r
　　where　qual;

意味　R リレーション R から、条件式 qual を満足する関係性組 r の集合を消去する。

```
R ← R − { r | r∈R ∧ qual(r, x1, ・・・, xm)
                ∧ x1 ∈ X1 ∧ ・・・ ∧ xm∈Xm};
```

更新伝播

i）タイプがTである全てのRの主属性p（atype(p)=T）に対して、pによって関係づけられる全てのE－リレーション組eを消去する。

```
sbegin
   T ← { r | r∈R ∧ qual(r). } ;
   for ∀r∈R do
          Rdelete(r, R)
send;
procedure    Rdelete(r, R) ;
sbegin
 for  ∀E∈ETs(R)   do
  for  ∀e∈E( r〔p〕 = e〔k〕 ∧ e−rel(p)=E )
   do
          Edelete(e, E) ;
 delR(r, R)
send of Rdelete;
```

図4.14に於て、Rから組rが消去されると、E1とE2内の関連する組e1とe2とが消去される。

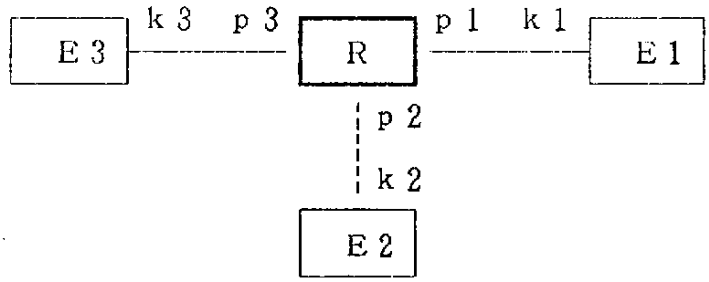


図 4.14

2－2）追　加

a）事象組の追加

構文

```
range      (e, E);
range      (x1, X1)・・・(xn, Xn) ;
append to  E(a1=aexp1, ・・・, am=aexpm )
     where qual;
```

意味

Eのスキームを$\underline{E}(\underline{k}, a1, \cdots, am)$とする（k＝patt(E)）．条件qualを満足する組e，x1，・・・，xnから生成される組e＝(a1＝$aexp_1$ ・・・，am＝$aexp_m$)を、Eに追加する。この時、eの主属性kの値は、データベース内で一意である様にシステムによって、自動生成される。

$$E \leftarrow E \cup \{ e \mid e=(k, aexp_1, \cdots, aexp_m) \wedge k=new\text{-}e\text{-}key \wedge qual(e, x1, \cdots, xn) \}$$

new-e-keyは、データベース内で一意なEキーの値を生成する関数である。

更新条件

Eと関連するRリレーションをRとする。この時、Rの主属性pが、e-rel(p)＝EでTタイプ(atype(p)＝T)の時、同時に、r(p)＝e〔k〕なるrを、rに追加せねばならない。このことは、同時に、すべてのRの主属性p'(p'≠p)に対して、

$$e'〔k'〕=〔p'〕 \wedge e' \varepsilon E' \wedge e\text{-}rel(p')=E'$$

なるe'が、E'内に存在せねばならないことである。この為、追加文appedのqualは、これらのe'を識別せねばならない。

eの主属性pの値を、セットしてはならない。

```
sbegin
   T ← { t | t=(a1, ・・・, am, pe1, ・・・, pel)∧
             a1=aexp1 ∧・・・∧ am=aexpm ∧
             pe1=e1〔k1〕∧・・・∧ pel=el〔kl〕∧
             e1εE1 ∧・・・∧ elεEl ∧
             qual(e1, ・・・, el, e) } ;
   for ∀tεT do
      e〔a1, ・・・, am〕 ← t〔a1, ・・・, am〕 ;
      e〔k〕 ← new-e-key ;
      for ∀RεRTs(E) do
       begin
        for ∀pεpatt(R) do
                r〔p〕 ← t〔pe*〕;
        appR (r, R)
       end
send;
```

ここで、pe*は、主属性pに対応したT内の属性である。

例えば、図4.15で、Eに組eを追加する為には、Eーリレーション E1、E2、E3の中の対応する組e1，e2，e3を識別し、Rーリレーション R1、R2に各々（e1、e2、e），（e3、e）を追加せねばならない。

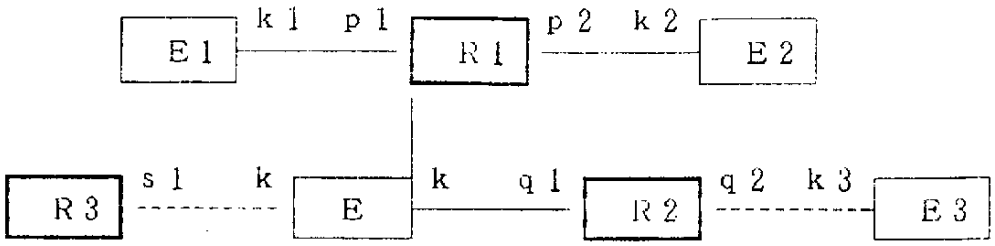


図 4.15

b）関係性組の追加

構文

range (r, R)(x1, X1)・・・(xm, Xm);
append to R(p1=patt$_1$, ・・・, pn=patt$_n$,
a1=aexp$_1$, ・・・, ak=aexp$_k$)
where qual;

意味

条件式qualを満足する関係性組rを、Rリレーション R（スキーム R（p1, ・・・, pn, a1, ・・・, am））に追加する。

R ← R ∪ { r | r=(patt$_1$, ・・・, patt$_n$,
aexp$_1$, ・・・, aexp$_n$) ∧
qual(r, x1, ・・・, xm) }

patt$_i$は、主属性の値又は、この値を取る変数である。

更新条件

i）Rが、タイプTで、かつ単一主キーである主属性pを持つならば、Rへの追加は出来ない。

ii）rが関係づける全ての事象組は、Rの関係づけるEリレーション内に存在しなければならない。

sbegin
T ← { r | r=(patt1, ・・・, pattn,
aexp$_1$, ・・・, aexp$_m$) ∧
qual(r, x1, ・・・, xm)};
for ∀t∈T do

appR （t, R）

send;

図4.1.5に於て、R2の主属性ql（atype（ql）=T）がR2の単一主キーの時、R2への追加は出来ない。

2－3）置　換

a）事象組の置換

構文

range　（e, E）（x1, X1）・・・（xn, Xn）;

replace　e（a1=aexp$_1$, ・・・, am=aexp$_m$）

where　qual;

意味

条件を満足するEリレーションE（スキームE（k, a1, ・・・, am）　k=patt（E））の事象組eの集合を、（e〔k〕、ea1, ・・・, eak）の集合で置換する。

$$E \leftarrow (E - \{ e \mid e \in E \wedge qual(e) \}) \cup \{ e' \mid e' = (e〔k〕, aexp_1, \cdots, aexp_m) \wedge qual(e) \}$$

更新条件

i）Eの主キーの値を置換えることは出来ない。

sbegin

$$T \leftarrow \{ e' \mid e' = (e〔k〕, aexp_1, \cdots, aexp_m) \wedge qual(e) \};$$

for　$\forall t \in T$　do

repE（e, E）

send;

b）関係性の置換

構文

range　（r, R）（x1, X1）・・・（xp, Xp）;

replace　r（p1=patt$_1$, ・・・, patto=pattn,

a1=aexp$_1$, ・・・, am=aexp$_m$）

where　qual;

意味

条件式を満足するRリレーションR内の関係性組rの集合を、（p1, ・・・, pn,

ｒａｌ，・・・，ｒａｋ）の集合によって置き換える。

$$R \leftarrow (R - \{ r \mid r \in R \wedge qual(r) \}) \cup \{ r' \mid r' = (patt_1, \cdots, patt_n, aexp_1, \cdots, aexp_m) \wedge qual(r) \}$$

更新条件

ｉ）Ｒのタイプ Ｔ，又は単一主キーの主属性ｐの値を置き換えてはならない。

ｉｉ）Ｅ＝ｅ－ｒｅｌ〔ｐ〕とすると、置換する前に、ｅ〔ｋ〕＝ｒ〔ｐ〕（ここで、ｐ＝ｐａｒｒ（Ｅ））なるｅがＥ内に存在せねばならない。

図4.15のＲ２の主属性ｑ２は、Ｔタイプなので、ｑ２の値の変更は出来ない。

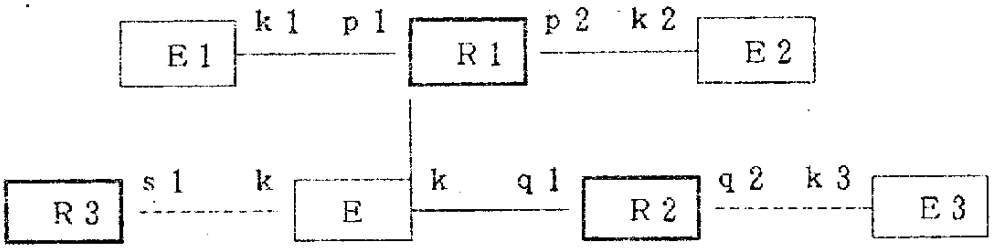


図4.15

4.3.3　変換アルブリズム

本項では、ＱＵＥＬ更新文（ａｐｐｅｎｄ，ｄｅｌｅｔｅ，ｒｅｐｌａｃｅ)からＣＯＢＯＬ ＤＭＬプログラムの生成について述べる。

Ａ　追　加（ａｐｐｅｎｄ）

追加（ａｐｐｅｎｄ）は、ＬＣＳリレーションに対して、新たな組の挿入を行なう。

```
range   (x1, X1) (x2, X2) ・・・(xm, Xm);
append to X (tl(t1, ・・・, te))
            where  qual(x1, ・・・, xm);
    ここで、ti ∈ {x1, ・・・, xm}
```

これは、リレーションＸに、ｑｕａｌを満足する組x_1，・・・，ｘｍから生成されるｔｌを追加することを意味している。

$$T \leftarrow \{t \mid t = tl(t_1, \cdots, te) \wedge qual(x_1, \cdots, xm) \wedge t_1 \in var(a) \wedge \cdots \wedge te \in var(a) \wedge x_1 \in X_1 \wedge \cdots \wedge xm \in Xm\};$$

$$X \leftarrow X \cup T;$$

ＬＣＳでは、リレーションに対して、１つの組（ｔｕｐｌｅ）を挿入するための条件が、スキーマ構造によって与えられる。例えば、ＣＯＤＡＳＹＬモデルのＭ／Ａセット型に対応して、子オカーランスをｓｔｏｒｅするためには親オカーランスがそれ以前に存在しなければならない。

ａｐｐｅｎｄ文の構造

構文：　append　to　<rel-name>　(<target-list>)
　　where　<qual>　;

ａ　ａｐｐｅｎｄ文の意味

(1)　Ｅリレーションへの組ｅの追加

Ｅリレーション（Ｅ）に組ｅを追加するためには次の条件が満足されなければならない。

∀Ｒ∈ＲＴｓ（Ｅ）に対して

$$e[k] = r[p]$$

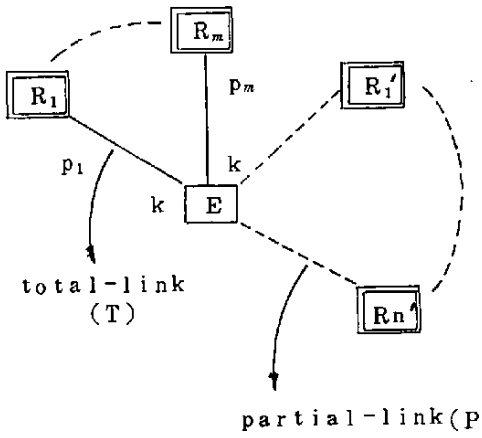
（ここで　$k = pkey(E) \wedge p \in patt(R) \wedge e\text{-}rel(p) = E$）

なる組ｒがｅと同時に、ｒに追加されなければならない。

例えば、右図においてｅをＥに追加するためには、R_1，・・・，Ｒｍに対して、各々

$$r_1[p_1] = e[k]$$
$$\vdots \qquad \vdots$$
$$rm[pm] = e[k]$$

なる組 r_1 , ・・・, rmも同時に追加される必要がある。



$$RTs(E)=\{Ri \mid i=1, \cdots, m\}$$

図 4.16

(2) Rリレーションに対する組rの追加

Rリレーション(R)に対して、組rを追加するためには、次の条件を満足しなければならない。

$\forall E \in Es(R)$ に対して

$$r[p] = e[k]$$

(ここで、$p \in patt(R) \wedge e\text{-}rel[p]=E \wedge k=pkey(E)$)

なる組eがEに存在しなければならない。

右図の例では、Rに組rを追加するためには、

E_1 , ・・・, Emが各々

$e_1[k_1]=r[p_1]$

⋮ ⋮

$en[kn]=r[pn]$

なる組 e_1 , ・・・, enが存在せねばならない。

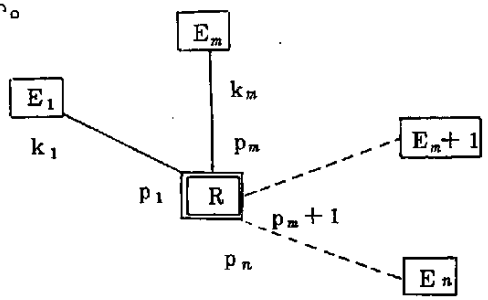


図4.17

b append文の処理概要

range (x_1 , X_1)(x_2 , X_2)・・・(xm, Xm) ;

a:append to X (a_1 =aexp_1 , ・・・, ak=aexp_k)

where qual(x_1 , x_2 , ・・・, xm) ;

上のappend文に対して以下を定義する。

var(a) ≜ $\{xi \mid i=1, 2, \cdots, m\}$

up－rel(a)≜ X

rat(a) ≜ {ai | i = 1, 2, ・・・, k}

tat(a) ≜ {ti, ai | aexp$_1$, ・・・, aexp$_k$ のどれかによって参照される属性}

tl(x) ≜ {x。a | x。a∈tat(a)}

≜ xについての目標属性

aexp(a) ≜ {aexp$_i$ | i = 1, 2, ・・・, k}

(1) XがEリレーションのとき

append aについて、次の条件が成りたたねばならない。

1) ∀b∈rat(a)に対して b≒patt (X)

即ち、目標リスト内の結果属性(a$_1$, ・・・, ak)は、どれもXの主属性(primary 属性)であってはならない。

2) ∀E ∈ TEs(X)に対して、append文aのqualはEを参照していなければならない。

ここで、TEs(X)は次のように定義される。

TEs(X)≜{X' | R∈RTs(X) ∧ X' ∈ Es(R) ∧ X' ≒ X}

また、RTs(E)は、Eと関連したRリレーションの中で関連性がtotalなものの集合である。

右図においては

TEs(E)={E$_1$, E$_2$, E$_3$, E$_4$, E$_5$}

となる。

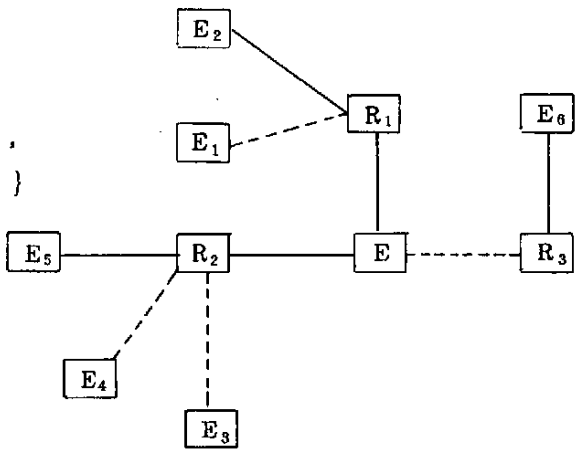


図4.1 8

ここで、TEs(X)={E1, E2, ・・・, En}とする。

Eiの組変数をeiとする。

tvar(a)={ei | i=1, 2, ・・・, n}

また、qual内の変数をvar(q)とするとき

nvar(a)≦var(q)−tvar(a)

={y1, ・・・, yp}

append aは次のようになる。

```
range    (e1, E1)・・・(en, En)
         (y1, Y1)・・・(yp, Yp);
a:append to X (a1=aexp1, ・・・, ak=aexpk)
        where qual (e1, ・・・, en, y1, ・・・, yp);
```

但し TEs(X)={Ei | i=1, 2, ・・・, n}

tvar(a)={ei | i=1, 2, ・・・, n}

nvar(a)={yj | j=1, 2, ・・・, p}

ei ∈ tvar(a)は、Xに追加される組(tuple)xの追加と同時に、eとxを関連づけるRリレーションRに組r

(r〔p1〕=e〔k1〕∧r〔p2〕=x〔k2〕∧e−rel(p1)=E∧e−rel(p2)=X)

を追加するために必要である。

3) E∈TEs(R)が複数の変数(e1、e2)を持ち、これがqualによって参照されているとき、どれがtvar(a)かを明らかにする必要がある。このとき、ユーザは、append文内で参照する変数の先頭に@を附して、tvarであることを示さなければならない。

例えば、e1がtvar(a)とすると

e1.a1=x.a5 and e2.a2=600 and ・・・

⇓

@e1.a1=x.a5 and e2.a2=600 and ・・・

(2) XがEリレーションのときの処理概要

1) まず、次の問合せを実行する。

```
range    (e1, E1)・・・(em, Em);
range    (y1, Y1)・・・(yp, Yp);
retrieve into T (t1=e1.@E1, ・・・, tm=em.@Em,
                 a1=aexp1, ・・・, ak=aexpk)
        where qual (e1, ・・・, em, y1, ・・・, yp);
```

2) Tが生成された後、それをもとに次のDMLを実行する。

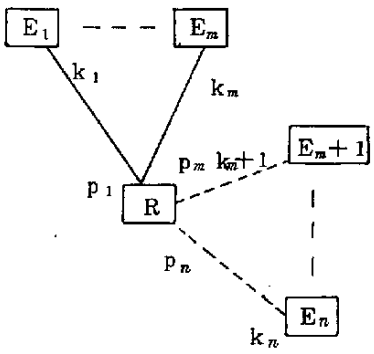
```
for ∀t ∈ T do
  begin;
        find E1 ; db-key is t[t1].
                  ⋮
        find Em; db-key is t[tm].
        X.a1 ← T.a1 ;
                  ⋮                    } 新しい組の値のセット
        X.ak ← T.ak;
        store X.
  end;
```

(3) XがRリレーションのとき

append文aについて、次の条件が成りたたねばならない。

1) ∀E ∈ Es(X)に対して、

$ai \in rat(a)\ (aexp_i = "e.k" \wedge k = pkey(E))$



左図において、aの目標リストは

{ p1=e1.k1, ・・・, pn=en.kn }

を含んでいなければならない。

図4.19

これは、Rリレーション(X)内に組xが存在することが可能なためには

$x[p_1] = e_1[k_1] \wedge \cdots \wedge x[pn] = en[pn]$

なる組e1, ・・・, enが各々E1, ・・・, Enになければならないためである。

即ち、目標リストは、xが関連づけるEリレーションの組eを識別する機能を有していなければならない。

(4) XがRリレーションのときの処理概要

1) XがM/A set-typeのとき (Mandatory/Automatic)

error

("X can not be appended")

2) XがO/M set-typeのとき (Optional/Manual)

i) range (e_1, E_1)(e_2, E_2);

range (y_1, Y_1)(y_2, Y_2)・・・(yp, Yp);

retrieve into T ($p_1 = e_1.@E_1$, $p_2 = e_2.@E_2$)

where qual (e_1, e_2, y_1, ・・・, yp);

ここで role(p_1)=OWNER ∧

role(p_2)=MEMBER ∧

erel(p_1)=E_1 ∧ erel(p_2)=E_2 とする。

ii) i)の問合せの結果Tをもとに、次の処理を行なう。

for ∀t(t∈T) do

begin;

find E_1; db-key is t[p_1].

find E_2; db-key is t[p_2].

connect E_2 to X.

end;

3) XがLink-record-typeのとき

i) まず次の問合せを実行する。

range (e_1, E_1)・・・(en, En);

range (y_1, Y_1)・・・(yp, Yp);

retrieve into T ($p_1 = e_1.@E_1$, ・・・, $pn = en.@En$,

$a_1 = aexp_1$, ・・・, $ak = aexp_k$)

where qual (e_1, ・・・, en, y_1, ・・・, yp);

ここで、e-rel(pi)=Ei

X(p_1, ・・・, pn, a_1, ・・・, ak)

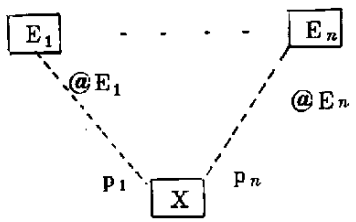


図4.20

ii) Tが生成された後、次の処理を行なう。

for ∀t∈T do

```
begin
        find  E1 ; db-key is t(p1)  ┐ 新レコード型のオカーランスのlocate
                  :                 │
        find  En ; db-key is t(pn)  ┘
        X.a1←T.a1 ;・・・X.ak←T.ak ;  } 新しい組の生成

        store X.

end ;
```

B 置 換 （replace）

置換（replace）は、リレーションの組を、新たな組によって置き換えることを行なう。

```
   range  (x, X);
   range  (x1 , X1 )・・・(xm , Xm);
r:replace    x  (tl(t1 , ・・・, tk))
       where   qual  (x, x1 , ・・・, xm );
     ここで  ti  ∈  {x, x1 , ・・・, xm }
```

replace文rは、qual（x, x_1 , ・・・, x_m ）を満足するx（x∈X）を、tl（t_1 , ・・・, t_k ）によって生成される組で置き換えることを表わしている。

a replace文の構文

構文： replace <var> (<target-list>)
where <qual> ;

b replace文の意味

(1) Eリレーションの組eのreplace

Eリレーションの組eを、新たな組e′で置換するためには次の条件を満足しなければならない。

e〔k〕=e′〔k〕

ここで、 k=pkey（E）（=patt(E)）

従って、Eリレーションでは、主キー属性の値を置き換えることはできない。

(2) replace文の処理概要

```
   range  (x, X) (x1 , X1 )・・・(xm , Xm);
r:replace   x  (a1 =aexp1 , ・・・, ak=aexpk )
       where  qual  (x, x1 , ・・・, xm );
```

上記のreplace文に対して以下を定義する。

var(r)　≙ {x, x_1, ・・・, x_m }

up−rel(r)≙　X ≙ updateされるリレーション

rat(r)　≙ {ai | i=1, ・・・, k }≙結果属性の集合

tat(r)　≙ {ti. ai | ti. aiはaexp$_1$ 〜aexp$_k$ のどれかによって参照される属性}

≙ 目標属性の集合

tl(α)　≙ {α. a | α. a∈tat(r) ∧ α∈var(r)}

≙ 変数αの参照する目標属性

aexp(r)　≙ {aexp$_i$ | i=1, ・・・, k}

att(X)　≙ Xの属性集合

prat(r)　≙ {a | a∈rat(r) ∧ a∈patt (X)}

prat(r)　≙ {a | a∈rat(r) ∧ a∉patt (X)}

(i) XがEリレーションのとき

replace　rについて次の条件が成りたたねばならない。

1) ∀a∈rat(r)に対してa≒patt (X)

即ち、結果属性として、primary属性があってはならない。

(ii) XがEリレーションのときの処理概要

1) 次の問合せを実行する。

range (x, X)(x_1, X_1)・・・(x_m, Xm);

q:retrieve into T (t$_1$ =x. @X,

a_1 =aexp$_1$, ・・・, ak=aexp$_k$)

where qual (x_1, ・・・, x_m);

ここで、

x. @Xはreplaceされるtupleを示している。

a_1, ・・・, a_kはreplaceするtupleの値を示している。

2) Tをもとにして、実際のreplaceを行なう。

for ∀t∈T do

begin;

find X;db−key is t〔t$_1$〕.

get X.

{内容のreplace}

X. a_1 ←T. a_1 ;・・・X. a_k ←T. a_k ;

modify X.

end;

(iii) XがRリレーションのとき

1) Xがセット型のとき

replace文は次の形をしている。

range (x, X) (x_1 , X_1) ・・・(x_m , Xm) ;

r : replace x (p_1 = aex p_1 , p_2 = aex p_2)

where qual (x, x_1 , ・・・, x_m) ;

ここで、p_1 , p_2 = pkey (X) である。

a) 条 件

rat (r) = { p_1 , p_2 } = patt (X)

aex p_1 = r. @R ここで e−rel (p_1) = R

aex p_2 = x. p_2

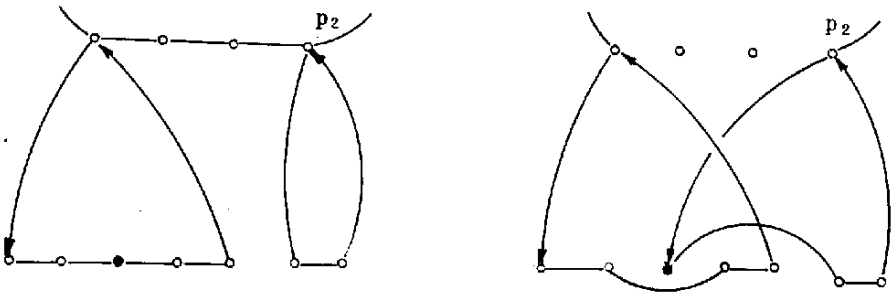


図 4. 2 1

この条件は、子レコードオカーランスが異なった親レコードオカーランスのセットトオカーランスに移ることだけができることを示している。

b) 処 理 概 要

1) Xの親レコード型、子レコード型を各々AとBとする。

AとBに対応したリレーションの組変数をa, bとする。

retrieve into T (p_1 = a. @A, p_2 = b. @B)

where qual (x, x_1 , ・・・, x_m) ;

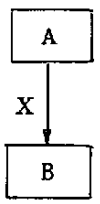


図 4.2 2

2) Tにもとづいて、modifyを行なう。

```
for ∀t∈T do
    begin
        find A;db-key is t〔p1〕 {新しい親オカーランス}
        find B;db-key is t〔p2〕 {子レコードオカーランス}
        modify B only X membership
    end;
```

3) Xがリンクレコード型のとき

```
    range (x, X)(x1, X1)・・・(xm, Xm);
r:replace x (p1=pexp1, ・・・, pk=pexpk,
                a1=aexp1, ・・・, ae=aexpe)
        where qual (x, x1, ・・・, xm);
```

ここで、prat(r)={ p_1, p_2, ・・・, p_k }

nrat(r)={ a_1, a_2, ・・・, a_e }である。

a) 条 件

prat(r)=φのとき

リレーションXのtuple xの主属性以外の値のreplaceであるから、Eリレーションのreplaceと同じである。

prat(r)≠φのとき

pi=pexp$_i$ の時 pexp$_i$=ri.@Ri (ここで、e rel(pi)=Ri)でなければならない。

即ち

```
replace x (p1=r1.@R1, ・・・, pk=rk.@Rk,
                a1=aexp1, ・・・, ae=aexpe)
        where qual (x, x1, ・・・, xm);
```

b) 処 理 概 要

1)

```
    range (x, X)(x1, X1)・・・(xm, Xm);
q:retrieve into T (t1=x.@X,
                p1=r1.@R1, ・・・, pk=rk.@Rk,
                a1=aexp1, ・・・, ae=aexpe)
        where qual(x, x1, ・・・, xm);
```

t_1 は、replaceされるXリレーションの組xをidentify

する。

主属性 $p_1, \cdots, p_k$ は、xが関連づける新たなEリレーションの組 $r_1, \cdots, r_k$ をidenfifyしている。

$a_1, \cdots, a_e$ は、xの新たな属性値である。

2)

```
for ∀t∈T do
begin
        find R1 ; db-key is t(p1)
                  ⋮
        find Rk ; db-key is t(pk)
        find X  ; db-key is t(t1)
        get X.
        x.a1 ← t.a1 ; ··· x.ae ← t.ae ;
        modify X only S1 , ···, Sk membership
end;
```

(ここで、Si＝role(pi))

C 消 去 (deletion)

消去は、リレーションから条件を満足するtupleの除去を行なう。

```
range (x, X)(x1 , X1 )···(xm , Xm );
delete  x  where  qual;
```

これは、条件(qual)を満足するリレーション(X)のtuple xの集合をXから除くことを表わしている。

1) $T \leftarrow \{x \mid x \in X \wedge qual(x, x_1, \cdots, x_m)\}$;

```
即ち、 range (x, X)(x1 , X1 )···(xm , Xm );
       retrieve into T (x.all)
                    where  qual;
```

2) $X \leftarrow X - T$;

LCSでは、1つのリレーションのtupleの消去は、このtupleの存在に依存している他のリレーションのtupleの消去をもたらすことがある。

この消去伝播は、CODASYL DBTGモデルでは、メンバシップクラスがMandatory／Automatic(M／A)のセット型を介した親から子レコードのオカーランスへの消去波及に対応している。

a　delete文の構文

構文：　delete　<var>　〔where　<qual>〕；

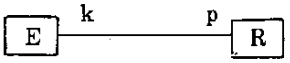
<var>は組変数（tuple　variable）

<qual>は、<var>を規定する条件式

b　delete文の意味

Eリレーション（E）とRリレーション（R）との間には、下の図に示すような2種のリンクがある。

a）total　link



b）partial　link

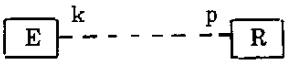


図4.23　全リンクと半リンク

a）は、

p∈patt（R）∧ e－rel（p）=E ∧ atype（p）=T

を示している。このとき、RとEの間には、次の関係が常に成り立たねばならない。

E〔k〕=R〔p〕

これは、次のことを意味する。

1）r∈Rが存在するためには、必ずe〔k〕=r〔p〕なるeがEに存在しなければならない。

2）同時に、e∈Eが存在するためには、必ず

e〔k〕=r〔p〕なるrがRに存在しなければならない。

従って、一方の消去は他方の消去をもたらすことになる。

b）は、

p∈patt（R）∧ e－rel（p）=E ∧ atype（p）=P

を示している。EとRは常に

E〔k〕 ⊇ R〔p〕 が成り立っている。

これは、a）の1）を表わしている。

(1)　Eリレーションの組eの消去

ある条件を満足するEリレーションの組（tuple）をeとする。eのdeleteはeと関連する全てのRリレーションの組のdeletionをもたらす。

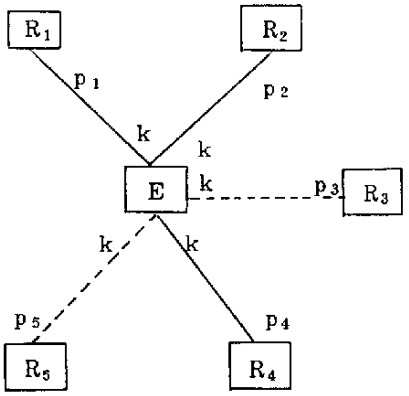


図 4.2 4

即ち

```
del  e;
        for  ∀R∈Rs(E)  do
                del  {r | r∈R  ∧  r〔p〕=e〔k〕
                        ∧  p∈patt(R)∧e−rel(p)=E};
```

例えば、図4.24では、Rs(E)={ R_1 , R_2 , R_3 , R_4 , R_5 }である。

このとき、e∈Eのdeleteは、Rs(E)内の全てのRiに対して

```
del  {ri  |  ri∈Ri  ∧  ri〔p〕=e〔k〕};
```

をもたらす。

(2) Rリレーションの組rの消去

Rリレーション(R)の条件を満足する組rのdeleteは、RからEへのlinkのタイプ(totalかpartial)によって異る。

a) total linkに対しては、r∈Rのdeletionは、r〔p〕=e〔k〕なるe∈Eのdeletionをもたらす。

```
del  r;
        for  ∀E∈ETs(R)  do
            del{e | e∈E  ∧  e〔k〕=r〔p〕  ∧
                    e−rel(p)=E  ∧  p∈patt(R)};
```

b) partial linkに対しては、r∈Rのdeletionは他に影響を及ぼさない。

(3) 消去の例

図4.25を例とする。この例において、$r_1 \in R_1$ なる r_1 をdeleteするものとする。

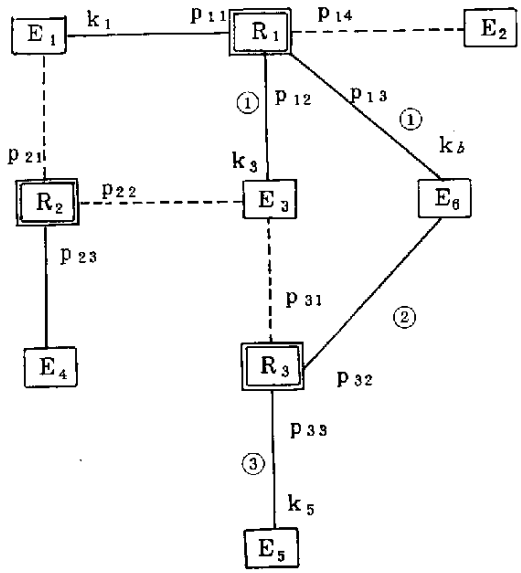


図4.2 5

del r_1 ; はBによって

① del{ e_1 | $e_1 \in E_1 \wedge e_1[k_1]=r_1[p_{12}]$ };

del{ e_3 | $e_3 \in E_3 \wedge e_3[k_3]=r_1[p_{12}]$ };

del{ e_6 | $e_6 \in E_6 \wedge e_6[k_6]=r_1[k_6]=r_1[p_{13}]$ };

更に、e_6 のdeletionはAによって

② del{ r_3 | $r_3 \in R_3 \wedge r_3[p_{32}]=e_6[k_6]$ };

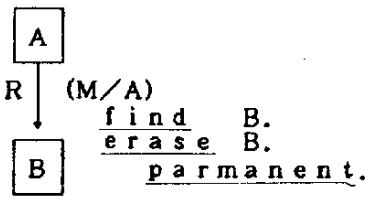
このr_3 のdeletionは

③ del{ e_5 | $e_5 \in E_5 \wedge e_5[k_5]=r_3[p_{33}]$ }; をもよらす。

QUEL	CODASYL DML
range (e, E) ; delete e where qual (e) ;	find E; erase E permanent.

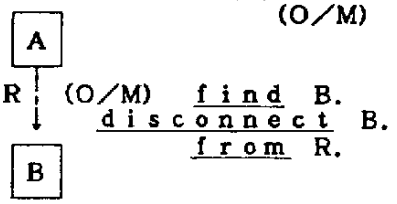
```
range (r, R) ;
  delete r
    where qual (r) ;
```

1) R は, set-type
   mem-class (R) = M/A

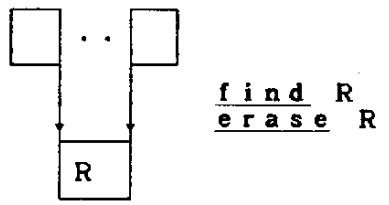


db-key (B) = r (B)

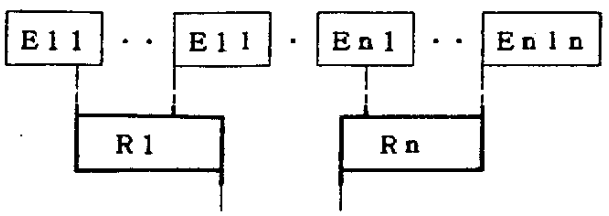
2) R は, set-type
   mem-class (R) = (O/M)



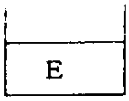
3)



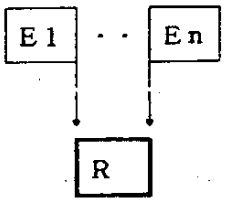
```
range (e, E) ;
range (e11, E11) ·· (en1n, En1n)
append to E (e')
  where qual (e11, ··, en1n, e) ;
```



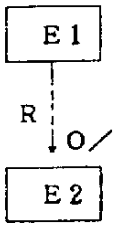

```
find E11.
...
find En1n.
e ← e' ;
store E.
```



```
range  (r, R) ;
range  (e1, E1) ..
       (en, En) ;
append to R (r')
  where
    qual (e1, ..., en, r) ;
```

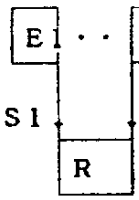


1)




```
find E1.
find E2.
connect E2
  to R.
```

2)

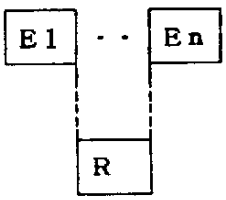



```
find E1.
...
find En.
store R.
```

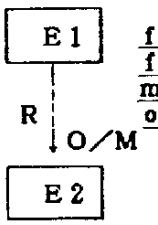
```
range  (r, R) ;
range  (e1, E1) ..
       (en, En) ;
replace r (r')
  where
    qual (e1, ..., en, r) ;
```

```
find E.
e ← e' ;
modify E.
```

```
range  (r, R) ;
range  (e1, E1) ..
       (en, En) ;
replace    r (r')
  where
    qual (e1, ..., en, r) ;
```



1) r (@E2) = r' (@E2)




```
find E1.
find E2.
modify R.
only R
membership
```

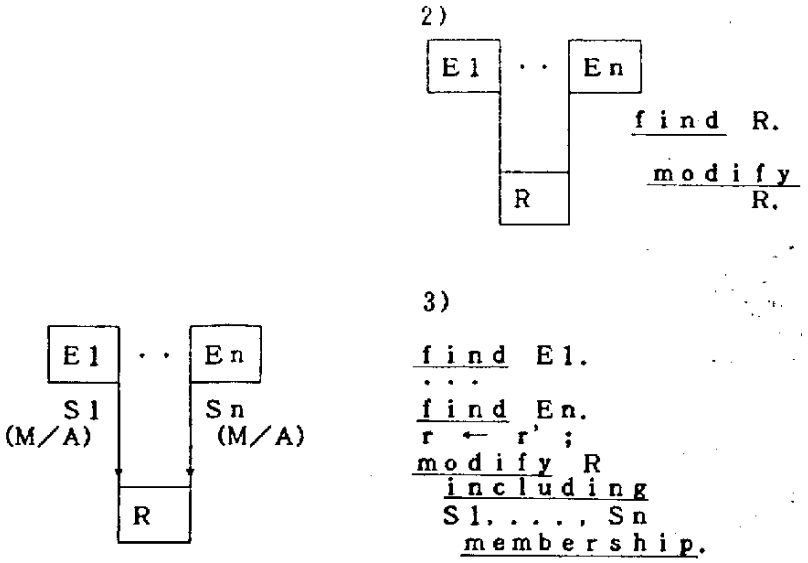


図 4.2 6　ＱＵＥＬとＤＭＬの対応

4.3.4　例

図 4.2 5にＬＤＰ－Ｖ２の使用例を示す。

プロジェクト番号が"ＭＭＵＩ８０"プロジェクトに属し、出身地が"ＴＯＫＩＯ"の従業員の出身地を"ＴＯＫＹＯ"に変更する。

まづ最初に、プロジェクト番号が"ＭＭＵＩ８０"に属する従業員の名前と出身地を検索して見る。

```
 READY
EXEC 'LDP0110.MMUI.CLIST(LDPV200)'

 *** JOBBS LDP-V2 START...  16.48.16.631

 #00010  RANGE (E,EMPLOYEE) (P,PROJECT);
 #00020  RANGE (PE,PROJ-EMP-LNK);
 #00030  RETRIEVE INTO RR (E.ENAME,E.BIRTH-PLACE,P.PNO)
 #00040  WHERE
 #00050        E.JE = PE.JE AND PE.JP = P.JP AND
 #00060        P.PNO="MMUI80";
 #00070  GO;

 RETRIEVE INTO RR ( ENAME=E.ENAME , BIRTH-PLACE=E.BIRTH-PLACE , PNO

      =P.PNO ) WHERE E.JE  = PE.JE  AND PE.JP  = P.JP  AND P.PNO  =

       "MMUI80" ;

     THE ELAPSE TIME FOR RETRVP.OM..     98
     THE ELAPSE TIME FOR ROGP.......   1010
     THE ELAPSE TIME FOR DOGP.......     21
```

```
    THE ELAPSE TIME FOR DFA1.....      6
    THE   ELAPSE TIME FOR DML.....       0
    THE ELAPSE TIME FOR DMLG.....   2501

    THE   ELAPSE TIME FOR  SORT....    7879
    THE ELAPSE TIME FOR DQGD..... 12501
KEQ56250I JOB DDB0110A SUBMITTED

* GBC TERMINATED
*   NUMBER OF FREE NODES  :   988
*   NUMBER OF USED NODES  :    13
#00080  STATUS;
KEQ56211I JOB DDB0110A EXECUTING
#00090  STATUS;
KEQ56213I JOB DDB0110A COMPLETED, WAITING FOR WRITER
#00100  OUTPUT;
KEQ56361I NO HELD OUTPUT FOR JOB DDB0110A ON HOLD CLASS
  RESLLT1...NPTNO=00000001
  RESULT1...NPTNO=00000002
 OUTPUT (001)
 DQGT001=MMUI80  TOKIO     TAKIZAWA
  RESLLT1...NPTNO=00000002
 OUTPUT (001)
 DQGT001=MMUI80  TOKIO     SUZUKI
  RESLLT1...NPTNO=00000002
 OUTPUT (001)
 DQGT001=MMUI80  TOKIO     YOKOTSUKA
  RESULT1...NPTNO=00000002
 OUTPUT (001)
 DQGT001=MMUI80  MIYAGI    KOSEKI
  RESULT1...NPTNO=00000002
 OUTPUT (001)
 DQGT001=MMUI80  TOKIO     TSUKAMOTO
 DQG==+00000001
          THE NUMB. OF LOCATED  GCC  = 00000011
          THE NUMB. OF ACCESSED GCC  = 00000010
          THE TIME FOR LOCATING      = 0000000020
          THE TIME FOR OUTPUT        = 0000000001
{TAKIZAWA  TOKIO     MMUI80
 SUZUKI    TOKIO     MMUI80
 YOKOTSUKA TOKIO     MMUI80
 KOSEKI    MIYAGI    MMUI80
 TSUKAMOTO TOKIO     MMUI80
 THE  CARDINALITY                = 00000005
 THE  NUMB.  OF  LOCATED  GCC    = 00000011
 THE  NUMB.  OF  ACCESSED GCC    = 00000010
 THE  NUMB.  OF  OUTPUT  GCC     = 00000006
 TOTAL TIME                      = 0000000140
 THE TIME FOR LOCATING           = 0000000020
 THE TIME FOR OUTPUT             = 0000000001
 THE TIME FOR MERGING            = 0000000047
  SETF-CONNECT = 000032500 SETF-I-CONNECT = 000009285
  SETF-CONNECT = 000006428 SETF-I-CONNECT = 000010000
  SETF-CONNECT = 000030000 SETF-I-CONNECT = 000010000
  SETF-CONNECT = 000044000 SETF-I-CONNECT = 000010000
  SETF-CONNECT = 000143570 SETF-I-CONNECT = 000010000
  SETF-CONNECT = 000024000 SETF-I-CONNECT = 000010000
  SETF-CONNECT = 000065000 SETF-I-CONNECT = 000010000
  SETF-CONNECT = 000074280 SETF-I-CONNECT = 000010000
```

検索結果

```
SETF-CONNECT = 000018928 SETF-I-CONNECT = 000010000
SETF-CONNECT = 000025000 SETF-I-CONNECT = 000010000
SETF-CONNECT = 000003571 SETF-I-CONNECT = 000010000
SETF-CONNECT = 000003571 SETF-I-CONNECT = 000010000
SETF-CONNECT = 000055000 SETF-I-CONNECT = 000010000
```

プロジェクト番号が"MMUI80"のプロジェクトに属し、出身地が"TOKIO"の従業員の出身地を"TOKYO"に変更する問合せは、次の様に実行される。

```
 READY
EXEC 'LDP0110.MMUI.CLIST(LDPV2GO)'

 *** JDDBS LDP-V2 START...  16.54.21.684

 #00010  RANGE (E,EMPLOYEE) (P,PROJECT);
 #00020  RANGE (PE.PROJ-EMP-LDP);
 #00030  REPLACE E (BIRTH-PLACE = "TOKYO" )              ← 置換コマンド
 #00040  WHERE
 #00050         E.@E = PE.@E AND PE.@P = P.@P AND
 #00060         P.PNO = "MMUI80"  AND E.BIRTH-PLACE = "TOKIO";
 #00070  GO;


 REPLACE TEMP0002 ( TEMPATT01="TOKYO" ) WHERE E.@E  = PE.@E  AND PE
      .@P  = P.@P  AND P.PNO  = "MMUI80"  AND E.BIRTH-PLACE  = "
      TOKIO" ;

     THE ELAPSE TIME FOR RETRVP,OM..   118
     THE ELAPSE TIME FOR RQGP.....  1157
     THE ELAPSE TIME FOR DQGP.....    24
     THE ELAPSE TIME FOR DFA1.....     4
     THE   ELAPSE TIME FOR DML.....      1
     THE ELAPSE TIME FOR DMLG.....  2085

     THE   ELAPSE TIME FOR  SORT....    7145
     THE ELAPSE TIME FOR DQGD..... 11940
 KEQ56250I JOB DDB0110A SUBMITTED

 * GBC TERMINATED
 *   NUMBER OF FREE NODES  :   986
 *   NUMBER OF USED NODES  :    15
 #00080  STATUS;
 KEQ56211I JOB DDB0110A EXECUTING
 #00090  STATUS;
 KEQ56211I JOB DDB0110A EXECUTING
 #00100  STATUS;
 KEQ56211I JOB DDB0110A EXECUTING
 #00110  STATUS;
 KEQ56211I JOB DDB0110A EXECUTING
 #00120  STATUS;
 KEQ56213I JOB DDB0110A COMPLETED, WAITING FOR WRITER
```

```
#00130  OUTPUT;
KE0563611 NO HELD OUTPUT FOR JOB DDB0110A ON HOLD CLASS
:    :% T1...NPTNO=00000001
 OUTPUT (001)
 DDGT001=
  RESULT1...NPTNO=00000001
 OUTPUT (001)
 DDGT001=
  RESULT1...NPTNO=00000001
 OUTPUT (001)
 DDGT001=
  RESULT1...NPTNO=00000001
 OUTPUT (001)
 DDGT001=
 DDG#=+00000001
          THE NUMB. OF LOCATED  OCC  = 00000011
          THE NUMB. OF ACCESSED OCC  = 00000004
          THE TIME FOR LOCATING      = 0000000024
          THE TIME FOR OUTPUT        = 0000000001
{TOKYO                                          }
{TOKYO                                          }  BIRTH-PLACE
{TOKYO                                          }  の値の置換
{TOKYO                                          }
 THE  CARDINALITY                  = 00000004
 THE  NUMB.  OF  LOCATED  OCC      = 00000011
 THE  NUMB.  OF  ACCESSED OCC      = 00000004
 THE  NUMB.  OF  OUTPUT  OCC       = 00000005
 TOTAL TIME                        = 0000000171
 THE TIME FOR LOCATING             = 0000000024
 THE TIME FOR OUTPUT               = 0000000001
 THE TIME FOR MERGING              = 0000000052
 TIME FOR UPDATE = 0000000022
  SETF-CONNECT = 000032500 SETF-I-CONNECT = 000009285
  SETF-CONNECT = 000006428 SETF-I-CONNECT = 000010000
  SETF-CONNECT = 000030000 SETF-I-CONNECT = 000010000
  SETF-CONNECT = 000044000 SETF-I-CONNECT = 000010000
  SETF-CONNECT = 000143570 SETF-I-CONNECT = 000010000
  SETF-CONNECT = 000024000 SETF-I-CONNECT = 000010000
  SETF-CONNECT = 000065000 SETF-I-CONNECT = 000010000
  SETF-CONNECT = 000074280 SETF-I-CONNECT = 000010000
  SETF-CONNECT = 000018928 SETF-I-CONNECT = 000010000
  SETF-CONNECT = 000025000 SETF-I-CONNECT = 000010000
  SETF-CONNECT = 000003571 SETF-I-CONNECT = 000010000
  SETF-CONNECT = 000003571 SETF-I-CONNECT = 000010000
  SETF-CONNECT = 000055000 SETF-I-CONNECT = 000010000
```

変更された結果を確認する為の検索は次の様に実行された。

```
 READY
EXEC 'LDP0110.MMUI.CLIST(LDPV2GO)'

 *** JDDBS LDP-V2 START...  16.58.24.606

 #00010  RANGE (E,EMPLOYEE) (P,PROJECT);
 #00020  RANGE (PE,PROJ-EMP-LNK);
 #00030  RETRIEVE INTO RRRR(E.ENAME,E.BIRTH-PLACE,P.PNO)
```

```
#00040  WHERE
#00050        E.@E = PE.@E AND PE.@P = P.@P AND
#00060        P.PNO = "MMUI80";
#00070  GO;


RETRIEVE INTO RRRR ( ENAME=E.ENAME , BIRTH-PLACE=E.BIRTH-PLACE ,

     PNO=P.PNO ) WHERE E.@E  = PE.@E  AND PE.@P  = P.@P  AND P.PNO

       = "MMUI80" ;

    THE ELAPSE TIME FOR RETRVP,GM..   100
    THE ELAPSE TIME FOR RGGP.......  1051
    THE ELAPSE TIME FOR DGGP.......  1941
    THE ELAPSE TIME FOR DFA1.......     5
    THE   ELAPSE TIME FOR DML......        1
    THE ELAPSE TIME FOR DMLG.......  1937

    THE   ELAPSE TIME FOR  SORT....    5770
    THE ELAPSE TIME FOR DGGD.......  7993
KEQ56250I JOB DDB0110A SUBMITTED

* GSC TERMINATED
*   NUMEER OF FREE NODES  :   988
*   NUMBER OF USED NODES  :    13
#00080  STATUS;
KEQ56211I JOB DDB0110A EXECUTING
#00090  STATUS;
KEQ56211I JOB DDB0110A EXECUTING
#00100  STATUS;
KEQ56213I JOB DDB0110A COMPLETED, WAITING FOR WRITER
#00110  OUTPUT;
KEQ56361I NO HELD OUTPUT FOR JOB DDB0110A ON HOLD CLASS
  RESULT1...NPTNO=00000001
  RESULT1...NPTNO=00000002
OUTPUT (001)
DGGT001=MMUI80  TOKYO     TAKIZAWA
 RESULT1...NPTNO=00000002
OUTPUT (001)
DGGT001=MMUI80  TOKYO     SUZUKI
 RESULT1...NPTNO=00000002
OUTPUT (001)
DGGT001=MMUI80  TOKYO     YOKOTSUKA
 RESULT1...NPTNO=00000002
OUTPUT (001)
DGGT001=MMUI80  MIYAGI    KOSEKI
 RESULT1...NPTNO=00000002
OUTPUT (001)
DGGT001=MMUI80  TOKYO     TSUKAMOTO
DGG#=+00000001
        THE NUMB. OF LOCATED  OCC  = 00000011
        THE NUMB. OF ACCESSED OCC  = 00000010
        THE TIME FOR LOCATING      = 0000000017
        THE TIME FOR OUTPUT        = 00000CCC01
TAKIZAWA  TOKYO     MMUI80
SUZUKI    TOKYO     MMUI80
YOKOTSUKA TOKYO     MMUI80
KOSEKI    MIYAGI    MMUI80
TSUKAMOTO TOKYO     MMUI80
```

置換後の検索結果

```
THE  CARDINALITY                    = 00000005
THE  NUMB.  OF  LOCATED  OCC        = 00000011
THE  NUMB.  OF  ACCESSED  OCC       = 00000010
THE  NUMB.  OF  OUTPUT  OCC         = 00000006
TOTAL TIME                          = 0000000149
THE TIME FOR LOCATING               = 0000000017
THE TIME FOR OUTPUT                 = 0000000001
THE TIME FOR MERGING                = 0000000048
 SETF-CONNECT = 000032500 SETF-I-CONNECT = 000009285
 SETF-CONNECT = 000006428 SETF-I-CONNECT = 000010000
 SETF-CONNECT = 000030000 SETF-I-CONNECT = 000010000
 SETF-CONNECT = 000044000 SETF-I-CONNECT = 000010000
 SETF-CONNECT = 000143570 SETF-I-CONNECT = 000010000
 SETF-CONNECT = 000024000 SETF-I-CONNECT = 000010000
 SETF-CONNECT = 000065000 SETF-I-CONNECT = 000010000
 SETF-CONNECT = 000074280 SETF-I-CONNECT = 000010000
 SETF-CONNECT = 000018928 SETF-I-CONNECT = 000010000
 SETF-CONNECT = 000025000 SETF-I-CONNECT = 000010000
 SETF-CONNECT = 000003571 SETF-I-CONNECT = 000010000
 SETF-CONNECT = 000003571 SETF-I-CONNECT = 000010000
 SETF-CONNECT = 000055000 SETF-I-CONNECT = 000010000
```

4.4　まとめと今後の課題

本章では更新用のＣＯＤＡＳＹＬスキーマから、ローカル概念スキーマ（ＬＣＳ）の生成と、ＬＣＳ　ＱＵＥＬ問合せからＣＯＢＯＬ　ＤＭＬプログラムの生成と実行について述べた。本システムは、ＬＤＰ－Ｖ２として、当協会のＭ－１７０ＦのＡＩＭ上に実現されている。

今後の課題としては、以下の点がある。

(i)　同時更新機能

(ii)　障害回復

(iii)　セキュリティ管理機構

5. まとめと今後の課題

本報告書では、オフィス情報システム（ＯＩＳ）を中心とした新たな情報システムに於いて中核となる分散型データベースシステム（ＤＤＢＳ）技術に対する技術的検討がまとめられている。本報告書で述べた新たな分散型データベースシステム（ＪＤＤＢＳ－Ⅱ）は、以下の特徴を有している。

(i) 物理的な放送通信機能を備えたローカルネットワークによる各種ＤＢＳの相互通信

(ii) 個人用データベースシステム、既存大型データベースシステム（特にＣＯＤＡＳＹＬ型のデータベースシステム）等の異種データベースシステムの統合

(iii) 従来の検索中心の静的なデータベース利用に加えて、データ更新を中心とした動的なデータベース利用のサポート

本年度は、昨年度モデル化したＪＤＤＢＳ－Ⅱの基本全体アーキテクチャ〔ＭＭＵＩ８１、ＴＡＫＩＭ８１ａ、ｂ〕を基にして、以下の検討及び一部モデルの実現と実験を行った。

(1) ローカルネットワークの放送通信（１：Ｎ通信）機能を用いた問合せの分散処理アルゴリズムの改良〔ＴＡＫＩＭ８２ｂ〕

(2) ＣＯＤＡＳＹＬ型のデータベースシステム（ＤＢＳ）に対する検索利用の為の高水準非手続的インタフェースＬＤＰ－Ｖ1.5〔昨年度実現〕の性能、機能、利用面からの総合的な評価と有用性の実証〔ＴＡＫＩＭ８１，ＴＡＫＩＭ８２ａ、ｅ〕

(3) (2)に加えて、ＣＯＤＡＳＹＬ　ＤＢＳに対する高水準非手続的な更新インタフェースシステムＬＤＰ－Ｖ２の実現〔ＴＡＫＩＭ８２ｄ〕

(4) ＤＤＢＳにおける分散更新処理方式の検討

ＪＤＤＢＳ－Ⅱの全体アーキテクチャ、(1)の分散問合せ処理、(4)の分散更新処理については、第２章において論じた。

この中で、以下の機構を用いた分散問合せ処理方式の提案を行った。

(i) ＤＢＳサイト間での通信処理の為のデータと制御情報の放送（１：Ｎ）転送

(ii) 半結合（ｓｅｍｉ－ｊｏｉｎ）の応答としてｂｉｔ－ｍａｐを用いることによる通信コストの低減

(iii) 選択度情報を用いない動的なスケジュール決定

(iv) 完全な分散制御

本分散問合せ処理方式によって、木問合せ〔ＢＥＲＮＰ７９〕に加えて、一般のネットワーク問合せを、結合属性の種類数に依存した回数の転送を行う事によって、有効な応答性とスループット（全処理時間）をもたらすことができる。

今後の課題としては、次の点がある。

(i) 複数ユーザ下での通信ネットワークと、各プロセッサ（ＤＢＳ）の競合現象の定量化（e.g. シミュレーション）

(ii) 障害に対する回復管理方法

(iii) 実現方法の検討

昨年度実現されたＣＯＤＡＳＹＬデータベースシステム（ＤＢＳ）に対する検索用の非手続的インタフェースシステムＬＤＰ－Ｖ１.５は、今年度以下の点について、その有効性を示す為に、総合評価を行った。

(i) 非手続的問合せ（ＱＵＥＬ）を処理する為の総合的なオーバヘッド（全処理時間）の定量化、

(ii) ＬＤＰ－Ｖ１.５の各機能モデュール毎のオーバヘッド（全処理時間）の定量化によるシステムのパフォーマンスボトルネックの明確化とその改良効果の測定

(iii) アクセスパス生成の為の最適化の為のオーバヘッドと、その効果（ｉ.ｅ. 生成されたＣＯＢＯＬ ＤＭＬプログラムの全実行時間）の関連性の明確化

(iv) 非手続的な言語ＱＵＥＬを用いて、ＬＤＰ－Ｖ１.５によって処理させることの利用面からの有効性を、従来のＣＯＢＯＬ ＤＭＬプログラムを作成し実行させる方法と比較して示す。

ＬＤＰ－Ｖ１.５の全処理時間は、問合せの種類にもよるが、ほぼ数秒から数１０秒によって必要なＣＯＢＯＬ ＤＭＬプログラムを生成できる。ＬＤＰ－Ｖ１.５内の各モデュールに於いて、大半の処理時間は、ＣＯＢＯＬ ＤＭＬプログラム（文字列）の生成と、入力されたＱＵＥＬ文字列の木表現への変換とその操作、グラフ構造の操作に費されている。

アクセスパスを見つける為の最適化には、全処理時間に対して高々数％が費やされるだけであった。最適化の評価では、次の点が明らかになった。

(a) 応答時間（全処理時間）は、アクセスされたレコードオカーランス数に依存している。

実際の物理的アクセスは、ページングに依存すると考えていたが、アクセスされるオカーランス数に比例する結果を得た。よって、アクセスされるオカーランス数を少なくすることによって、全処理時間を短縮できる。

(b) アクセスパスを見つける為に、サーチするレベルを深めても、ＬＤＰ－Ｖ１.５全体の処理時間に対する割合は高々数％であった。従って、ノード数が少い場合には、全ての可能なパスをサーチする方法も考え得る。

(c) 選択度（ｓｅｌｅｃｔｉｖｉｔｙ）、結合度（ｃｏｎｎｅｃｔｉｖｉｔｙ）といった統計情報に基づいたアクセスされるオカーランス数の見積りと、実際の結果では、我々のデータベース（ＰＲＤＢＳ）と問合せに対しては、相関がなかった。この理由は、各アプリケーションが用いるデータベースの値（オカーランス）の分布と、データベース全体の分布とが一致していない為と考えられる。この為、統計情報を各アプリケーション毎に保持すると共に、これらを実際のアクセスによるｄｙｎａｍｉｃ ｔｕｎｉｎｇによって実際の分布をよく反映するようにできた。

(d) 同一の問題に対しては、ＱＵＥＬによる入力、実行の全処理時間は数分～数１０分であるのに対して、ＣＯＢＯＬ ＤＭＬによっては数時間～数日を要した。この意味で、ＬＤＰ－Ｖ１.５を用いることによって、ＣＯＤＡＳＹＬ ＤＢＳへのアプリケーションソフトウェアの生産性を大幅に向上させ得る。

検索用の非手続的インタフェースＬＤＰ－Ｖ１.５に、更新機能を付加させたＬＤＰ－Ｖ２の実現を行った。検索に加えて、更新をサポートする為には、新たに以下の検討が必要になる。

(i) ＣＯＤＡＳＹＬスキーマ（Ｃ）と等価な非手続的なローカル概念スキーマ（ＬＣＳ）の生成

(ii) ＬＣＳに対する非手続的更新演算（ＱＵＥＬ）から、ＣＯＤＡＳＹＬスキーマに基づいた
ＣＯＢＯＬ　ＤＭＬの生成

(iii) 及びこれの実行

(i)では、互いの情報が保存されると共に、互いの更新の効果が等しくなければならない。この為、リレーショナルモデルを拡張したＬＣＳモデルを提案している。このモデルでは、ＣＯＤＡＳＹＬモデルに於ける更新の波及、入力条件等の更新依存性を表すことができる。

(ii)は、ＱＵＥＬに対して対応する演算を生成することによって処理できる。

(iii)では、同時更新を今回は考えない事にしている。この時、データベースの障害、ユーザによる誤った更新からデータベースを守る為に、ある時点で、コンシスタントなデータベースを、バックアップ（ディスク）にコピーすることにしている。今後、ログによるリカバリ、セキュリティの管理、同時更新機能の付加が必要になる。

ユーザから見た問題点としては、ユーザインタフェース言語ＱＵＥＬの利用性がある。ＱＵＥＬでは、一般ユーザにとって、結合式の入力、組変数の利用方法が難かしい。この為、図形、関数形式等のインタフェースの検討が必要である。

参 照 文 献

〔ADIBM 80〕
Adiba, M., Andrade J.M., Fernandez, F., and Toan, N.G.,
"An Overview of the Polypheme Distributed Database Management System,"
Proc. of the IFIP' 80, Tokyo and Melbourne, Oct. 1980, pp. 475—479.

〔BANCF 79〕
Bancilhon, F., "Supporting View Updates in Network Schemas",
Proc. of SIGMOD Conf., Boston, May, 1979, pp. 179—190.

〔BERNP 79〕
Bernstein, P. A., and Goodman, N.,
"Full Reducers for Relational Queries, Using Multi-Attribute Seme-Joins,"
Proc. of the IEEE Computer Networking Symposium, Gaithersburg, Maryland, Dec. 1979, pp. 206—215.

〔BERNP 80a〕
Bernstein, P. A., Shipman, D.W., and Rothnie, J.B.,
"Concurrency Control in a System for Distributed Databases (SDD—1),"
ACM TODS, Vol. 5, No. 1, Mar. 1980, pp. 18—51.

〔BERNP 80b〕
Bernstein, P. A., and Shipman, D.W.,
"The Correctness of Concurrency Control Mechanisms in a System for Distributed Databases (SDD—1),"
ACM TODS, Vol. 5, No. 1, Mar. 1980, pp. 52—68.

〔BERNP 80c〕
Bernstein, P. A., and Goodman, N.,
"Timestamp-Based Algorithms for Concurrency Control in Distributed Database Systems,"
Proc. of the 6th VLDB, Montreal, Oct. 1980, pp. 285—300.

〔BERNP 81a〕
Bernstein, P. A., and Chiu, D.W.,
"Using Semi-Joins to Solve Relational Queries,"
ACM Vol. 28, No. 1, Jan. 1981, pp. 25—40.

〔BERNP 81b〕
Bernstein, P. A., and Goodman, N.,
"Concurrency Control in Distributed Database Systems,"
ACM Computing Surveys, Vol. 13, No. 2, June, 1981, pp. 185—221
〔CODAS 73〕
CODASYL, "CODASYL Data Description Language Jounnal of Development,"June. 1973, NBS Handbook 113(1974)
〔CODDE 70〕
Codd, E. F.,"A Relational Model of Data fo Large Shared Data Bank,"
CACM, Vol. 13, No. 6, June, 1970, pp. 337—387.
〔DATEC 79〕
Data, "Looking and Recovery in a Shared Database System : An Application Programmer Tutorial,"Proc. of the 5th VLDB, Brazil, Oct. 1975, pp. 1—15.
〔DAYAU 78〕
Dayal, U., et. al, "On the updatability of Relational views,"
Proc. of the 4th VLDB, Berlin, Sept. 1978, pp. 368—377
〔ESWAK 76〕
Eswaran, K. P., et. al, "The Notions of Consistency and Predicate Locks in a Database System, "CACM, Vol. 19, Nov. 1976, pp. 624—633
〔EPSTR 78〕
EpsteingR. et. al, Distributed Query Processing in a Relational Database System," UCB/ERL M78/18, UC. Berkeley, April 1978.
〔FERNE 81〕
Fernandez, E. B, "Database Security & Integrity," Addison-Wesley, 1981.
〔GOODN 79〕
Goodman, N., Bernstein, P. A., Wong, E., Reeve, C. L., and Rothnie, J. B.,
"Query Processing in SDD—1 : A SYSTEM for Distributed Databases,"
CCA—79—06, CCA, Cambridge MA. Oct. 1, 1979, pp. 1—57.

〔GRAYJ 75〕
Gray, J., et. al, "Granulanity of Locks in a Shared Database,"
Proc. of the 1st VLDS, 1975, pp. 428—451
〔GRAYJ 79〕
Gray, J. N.,
"Notes on Data Base Operating Systems," Operating Systems
(Bayer, R. et al, eds.), Springer-Verlag, 1979, pp. 393-481.
〔GRAYJ 81〕
Gray, J., et al.,
"The Recovery Manager of the System R Database Manager,"
ACM Computing Surveys, Vol. 13, No. 2, June, 1981, pp. 223—243.
〔GUODM 81〕
Gouda, M. G., and Dayal, U.,
"Optimal Semijoin Schedules for Query Processing in Local
Distributed Database,"
ACM SIGMOD '81, Ann Arbor, Michigan, April-May 1981, pp.
164—174.
〔HEVNA 78〕
Hevner, A. R., and Yao, S. B.,
"Query Processing on a Distributed Database,"
Proc. 3rd Berkeley Workshop on Distributed Data Management
and Computer Networks, San Francisco, California, Aug. 1978,
pp. 91—107.
〔JDDBS 80〕
"分散型リソース処理技術の研究開発—分散型データベースシステム," 54—S—001, (財)
日本情報処理開発協会, Mar. 1980
〔KOHLW 81〕
Kohler, W. H.,
"A Survey of Techniques for Synchronization and Recovery in
Decentralized Computer Systems,"
ACM Computing Surveys, Vol. 13, No. 2, June, 1981, pp. 149—183.
〔LEBIJ 80〕
Le Bihan, J., Esculier, C., Lann, G. L., Litwin, W.,
Gardarin, G., Sedilot, S., and Treille, L.,
"Sirus:A French Nationwide Project on Distributed Data Bases,"

Proc. of the 6th VLDB, Oct. 1980

〔LINDB 80〕

Lindsay, et. al, "SINGLE AND MULTI−SITE RECOVERY FACILITIES," Distributed Data Bases(Dnaffen, L.W. Posle, Feds), Cambnidge Llniv. Press, Cambridge, 1980

〔MENAD 78〕

Menase, D. A., et. al, "Locking and Deadlock Detection in Distribuled Database," Proc. of the 3nd Benkeley Workshop, Aug. 1978, pp. 215−232

〔MMUI 81〕

"マンマシンユーザインタフェースに関する調査研究"

Jipdec55 SOO2, Mar. 1981

〔MOORR 79b〕

Moorc, R. C., "Handling Complex Queries in a Distributed Database," SRI Technical Note 170, 1,979.

〔ROWEL 79〕

Rowe, L. A., et. al, "Data Abstractions, Views and Updates in RIGEL," Proc. of ACM SIGMOD, Boston, May. 1979, pp. 71−81.

〔SMITJ 81〕

Smith, J. M., et. al, "Multibase−Integrating Heterogeneous Distnibuted Database Systems," AFIPS Conf. Proc., Vol. 50, pp. 487−497

〔SPYPN 80a〕

Spyratos, N., "TRASLATION STRUCTURES OF RELATIONAL VIEWS," Proc. of the 6th VLDB, Mortreol. Canada, Oct. 1980.

〔STONM 75〕

Stonebraker, M.,

"Implementation of Integrity Constraints and Views by Query Modification,"

Proc. International Conf. on Management of Data, May 1975, pp. 65−78.

〔STONM 76〕

Stonebraken, M., et. al, "The Design and Implementation of

INGRES," ACM TODS, Vol. 1, No. 3, 1976, pp. 189—222

〔TAKIM 78〕

Takizawa, M., et. al, "Resource Integration and Data Sharing on Heterogeous Resource Sharing," Proc. ICCC' 78, Kyoto, Sept. 1978

〔TAKIM 79〕

Takizawa, M. and Hamanaka, E., "The Four-Scheme Concept as the Gross Architectune of Distributed Database and Hetenogeneity Problems," JIP of IPSJ, Vol. 2, No. 3, Nov. 1979, pp. 134—142

〔TAKIM 80〕

Takizawa, M. and Hamanaka, E., "Query Translation in Distributed Database," Proc. of the IFIP' 80, Tokyo-Melbourne, Oct. 1980, pp. 451—456.

〔TAKIM 81a〕

Takizawa, M., "分散型データベースシステム JDDBS－Ⅱ－スキーマ層モデルと通信処理モデル"

京大数理解析研 Feb. 1981

〔TAKIM 81b〕

Takizawa, M.,

"分散型データベースシステムJDDBS－Ⅱと通信処理"

信号会オートマトン研 1981

〔TAKIM 82a〕

Takizawa, M., "Distribution problems in Distributed Database-Integration and Query Decomposition,"

to appear in JIP (of IPSJ), 1982

〔TAKIM 82b〕

Takizawa, M., "Distributed Query Processing Algoritnm in Local Broadcast Netwoks,"

Jipdec TR 82/03, Feb. 1982

〔TAKIM 82c〕

Takizawa, M.,

"Relational Interface System to CODASYL Database System in Heterogeneous Distributed Database Systems,"

Jipdec TR 82/04 Feb. 1982

〔TAKIM 82d〕
Takizawa, M., Yokotsuka, M., and Suzuki, M.,
"LDP−V 1.5の総合評価について"
京大数理解析研 Feb.1982
〔THOMR 79〕
Thomas, R. H.,
"A Majority Consensus Approach to Concurrency Control for Multiple Copy Databases,"
ACM TOAS, Vol. 4, No. 2, June 1979, pp. 180−209.
〔TOANN 79〕
Toan. N, et al, "A Unified Method for Query Decomposition and Shared Informetion Updating in Database Systems," Distr. Comp. Systems, Oct. 1979, pp. 679−685.
〔TOANN 81〕
Toan, N.G.,
"Distributed Query Management for a Local Network Database System,"
Proc. of the 2nd Intl. Conf. on Distributed Comp. Systems, Paris, France, April, 1981, pp. 188−196.
〔VERHJ 78〕
Verhofstad, J. S. M.,
"Recovery Techniques for Database Systems,"
ACM Computing Survey, Vol. 10, No. 2, June 1978, pp. 167−195
〔VINED 81〕
Vines, D. H. Jr.,
"A Dataflow Solution for Implementing Distributed Queries,"
5th, Berkeley, Feb. 1981, pp. 14−37.
〔WONGE 77〕
Wong, E.,
"Retrieving Dispersed Data from SDD-1 : A System for Distributed Databases,"
Proc. 2nd Berkeley Workshop on Distributed Data Management and Computer Networks, Berkeley, May 1977, pp. 217−235

〔WONGE 79〕
Wong, E., et. al, "Logical Design and Schema Conversion for Relational and DBTG Databases, "Proc. of E-R Cont., pp. 311-321, 1979
〔ZANIC 79a〕
Zaniolo, C., "Design of Relational Views over Network Schemas," Proc. of SIGMOD Cont., Boston, May. 1979, pp. 179-190

記号の説明

AB	ATの枝集合（ABR）
accn(l, y)	評価関数
acc1(l, y)	1レベルのサーチ評価関数
acc2(l, y)	2レベルのサーチ評価関数
acc3(l, y)	3レベルのサーチ評価関数
acn(S)	セットSの子オカーランスをアクセスする平均オカーランス数
ADBS	ACOS700のCODASYL型DBS
AIM	M−160, M−170FのCODASYL型DBS
AN	ATの節点集合
APG	アクセスパス生成
AT	アクセス木
BM	ビットマップ
CALC	calculation key 計算キー
CALC−item(R)	レコード型RのCALC項目
card(A)	集合Aのカーディナリティ
CE	CQGの辺集合
child(a)	アクセス木の節点aの子節点集合
ch(o)	出力木（OPT）の節点Oの子節点集合
ci	同一組変数を参照する述語の和
cond(a)	オカーランスa の条件述語
CN	CQGの節点集合
cnt(S)	セットオカーランス集合Sの平均結合度
cptr(bi)	アクセス木の節点biの一番左の子節点
current	実行単位（run−unit）の現在子（currency）
current(A)	エリアAの現在子
current(R)	レコードオカーランス集合Rの現在子
current(S)	セットオカーランス集合Sの現在子
CQ	CODASYL問合せ
CQG	CODASYL問合せグラフ
CQGD	CQGの分割
DBTGD	CODASYLスキーマ記述
DBTGDL	CODASYLスキーマ記述言語
DBキー	database key データベースキー
DEF(Vi)	ビュー定義

DFA	Depth first アルゴリズム
Di	domain値の集合(定義域)
di	Di内の要素(di∈Di)
DML	CODASYLモデルのデータアクセス言語
DMLD	DML記述
DMLG	DML生成
DMLI	DML情報
DNA−item(S)	セット型SのDNA項目
dom(ti)	データ項目tiの定義域
E(R)	レコード型Rに対応するLCSリレーション
FNS	自由節点空間(Free Node Space)
	Q−treeやATの節点がここに作られる。
frtrn(bi)	アクセス木の節点biの失敗復帰先
HI	異種性情報
HIP	スキーマ変換プロセッサ
HS	隠れ構造
item(R)	レコード型Rのデータ項目
iacn(S)	セットSの子B内のオカーランスbからその親aを見つける為にアクセスされるBの平均オカーランス数
IPF	会話型プログラミングファシリティ
ITP	対話型プロトコル
JBM	結合ビットマップ
jc_i (x, y)	主結合述語あるいは非主結合述語の和
jf(x, y)	結合式jci(x, y)の積
jp(x, y)	節点x, y間の結合述語
L	LCSリレーション
l	LCSリレーションに対する組変数
l−next(bi)	アクセス木の節点biの先祖の中で、最も近くにいるMember節点
LCS	ローカル概念スキーマ
LCP	ローカル通信プロセッサ
LCSG	ローカル概念スキーマ生成
LIS	ローカル内部スキーマ
LISG	ローカル内部スキーマ生成
MA	メンバシップクラス=Mandatory Automatic

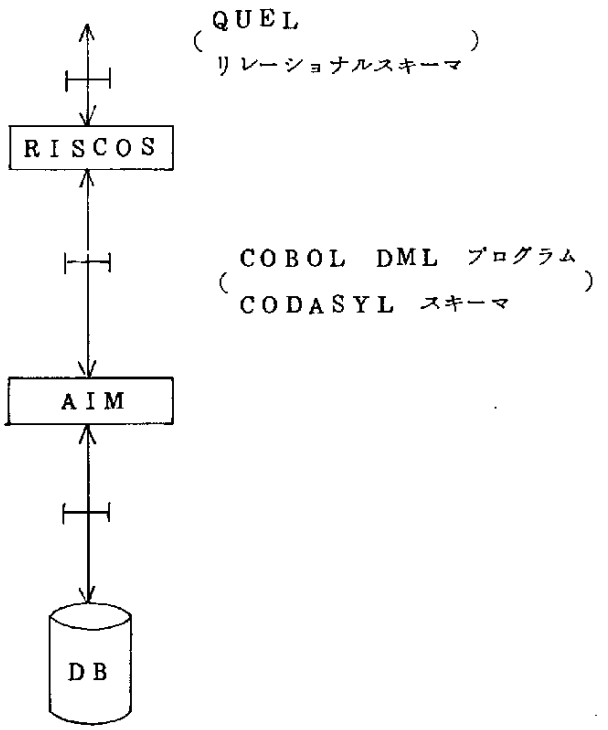
n e x t (b i)	アクセス木の節点ｂｉの次にアクセスする節点
n e x t (S)	セットオカーランスSのNEXTレコード
n j p (x , y)	節点x，y間の非主結合述語
NOC (T)	アクセス木Tのアクセスオカーランス数
n－p t r (S , r)	NEXTポインタの指すレコードオカーランス
n r f (a)	非主制限述語の積
n s p (x , y)	CQG節点x，y間の非主結合述語
nst_a	n r f (a)の選択度
OBR	出力木の枝集合
ON	出力木の節点集合
OM	メンバシップクラス＝OPTIONL MANUAL
OPT	出力木
o－p t r (S , r)	o w n e r ポインタの指すレコードオカーランス
o w n e r (S)	セット型Sの親レコード型
p a r (b i)	アクセス木の節点ｂｉの親節点
p a t t (R)	リレーションRの主属性集合
PJLa	LQG節点aを含む主結合辺集合
pjl_{ij}	主結合辺
p j p (x , y)	LQG節点x，y間の主結合述語
p－p t r (S , r)	PRIORポインタの指すレコードオカーランス
PRDBS	プロジェクト管理データベースシステム
p r f (S)	主制限述語の積
p r i o r (S)	セットオカーランスSのｐｒｉｏｒレコード
p s t a	p r f (a)の選択度
QM	問合せ変形
QTP	問合せ変換プロセッサ
Q－t r e e	QUEL問合せの内部表現
＜q u a l＞	条件式
RBE	リモートバッチエントリシステム（ACOS－７００）
RBM	制限ビットマップ
rc_i	制限述語ｒｐ（ｘ）の和
RES	リモートエントリシステム（M－１７０F）
r f (x)	xの制限式rc_i（ｘ）の積
R (L)	リンクレコード型に対応するＬＣＳリレーション
r p (x)	xの制限述語

RRT	参照木
R(S)	セット型に対応するLCSリレーション
sjp	セット結合述語
SO(r_i)	親レコードオカーランスr_iに対するセットオカーランス集合
sort−item(S)	セットオカーランスSのソート項目
sort−type(S)	セットオカーランスSのソート型
	(昇順(Ascending)又は降順(Descending)
sptr(b_i)	アクセス木の節点b_iの右隣りの兄弟節点
srtrn(b_i)	アクセス木の節点b_iの成功復帰先
t_i	データ項目
tl(x)	xの目標属性集合
tst$_T$	部分木Tの選択度
VMI	ビュー管理情報

付記1　ＲＩＳＣＯＳ（Ｒｅｌａｔｉｏｎａｌ　Ｉｎｔｅｒｆａｃｅ　Ｓｙｓｔｅｍ　ｏｖｅｒ　ｔｈｅ　Ｃｏｄａｓｙ　ｄａｔａｂａｓｅ　Ｓｙｓｔｅｍ）

1.1　ＲＩＳＣＯＳの概要

ＲＩＳＣＯＳは、ＡＩＭ　ＤＢＳに対するリレーショナル検索非手続的インタフェースシステムである。ＲＩＳＣＯＳは、ＬＤＰ－Ｖ1.5に対してユーザインタフェース機能を強化したシステムである。生成されたＣＯＢＯＬ　ＤＭＬプログラムの実行と実行結果のユーザへの出力を、メーカーから提供されたＩＰＦ（会話処理プログラミングランゲージ）を使用してローカル通信機能のオーバヘッドを避けている。



ＲＩＳＣＯＳは、

1）エンドユーザ向けの非手続的検索インタフェースである。非定型的業務に利用されるインタフェースである。

2）データベースを検索するＣＯＢＯＬ　ＤＭＬプログラムの開発支援ツールである。

1.2　ＲＩＳＣＯＳの機能

ＲＩＳＣＯＳの機能としては、次の様な機能を持つ。

1）ＣＯＤＡＳＹＬスキーマからリレーショナルスキーマの生成

2) リレーショナル問合せ（ＱＵＥＬ問合せ）から、ＣＯＢＯＬ　ＤＭＬプログラムの生成。

3) ＣＯＢＯＬ　ＤＭＬプログラムの実行と検索結果のユーザへの出力。

この３つの機能のうち、１）と２）はＬＤＰ－Ｖ1.5の機能と同じである。３）については、ＬＤＰ－Ｖ1.5のローカル通信機能を使用せず、ＩＰＦを利用した処理を行う。ＩＰＦを利用して、次の機能を実現する。

・ＣＯＢＯＬ　ＤＭＬ　プログラムの実行

・検索結果の出力

・実行ジョブの状態表示

・実行ジョブのキャンセル

・検索結果のセーブ

ＲＩＳＣＯＳは、

1) エンドユーザ向けの非手続的検索インタフェースである。非定型的業務に利用されるインタフェースである。

2) ＣＯＢＯＬ　ＤＭＬ　プログラムの開発支援ツールである。

1.3　システム構成

ＲＩＳＣＯＳは、スキーマ変換と問合せ変換とか成る。ＲＩＳＣＯＳのシステム構成は、図の様になっている。

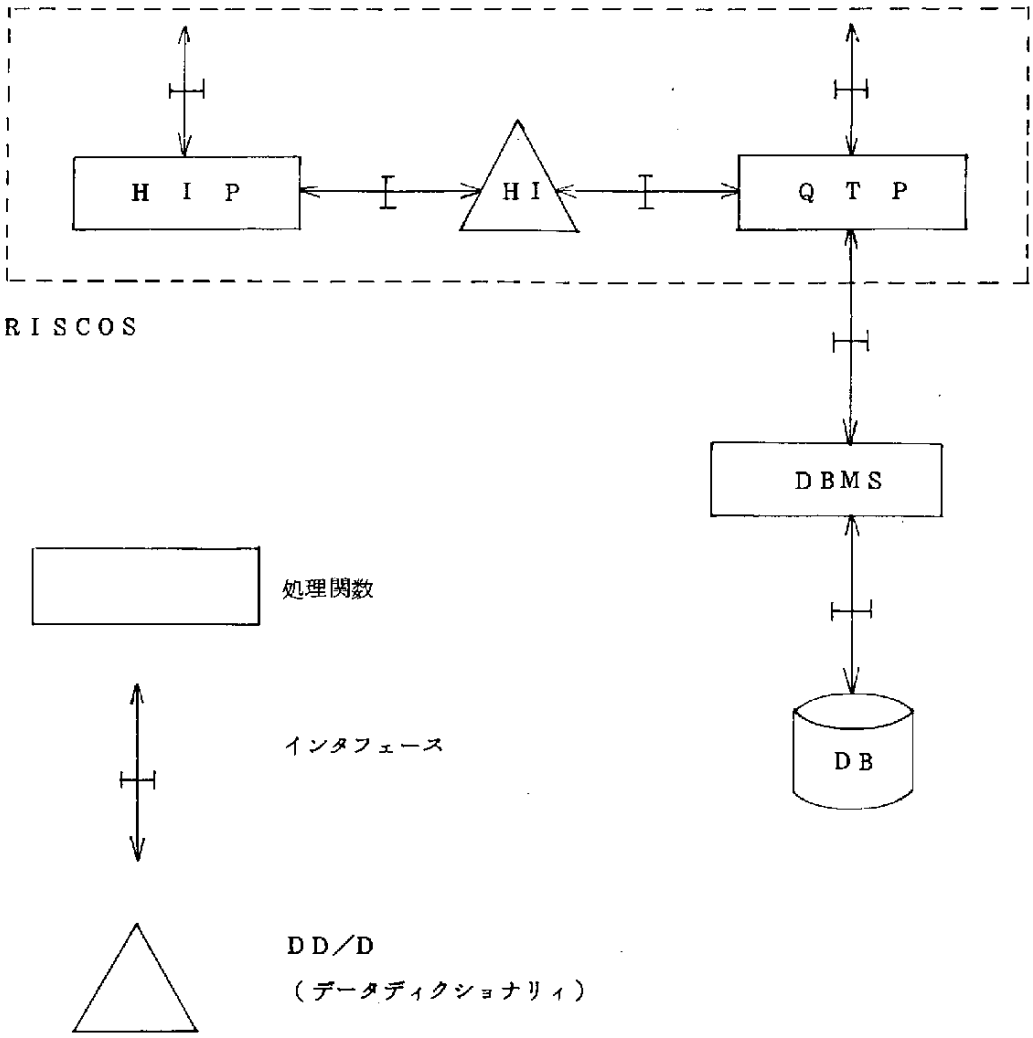


図　RISCOSのシステム構成図

1.4　ユーザインタフェース

RISCOSでは、スキーマ変換のインタフェースとしてはCODASYLスキーマ定義言語(DBTGDL)を、問合せ変換のインタフェースとしてはQUEL言語をサポートする。

(1)　CODASYLスキーマ定義言語

3.4節のCODASYLスキーマ記述を参照。

B　QUEL言語

a) RANGE文

<range-statement>::=RANGE (<var-list> <rel-name>)
{(<var-list> <rel-name>)};

<var-list>::= <var> | (<var> {,<var> })

<rel-name>::= <LCS-relation> | <view> | <snapshot>

RANGE文は、リレーション(<rel-name>)に対して、組変数又は、組変数の集合(<var-list>)を定義するものである。

例えば、リレーションEMPに対して組変数e1, e2を、リレーションSALに対してS1を定義すると

RANGE((e1, e2)EMP)(S1 SAL); となる。これは、

RANGE(e1 EMP)(e2 EMP)(S1 SAL); と等価である。

組変数を定義するリレーションはLCSリレーションでも、viewでも導出されたリレーション(<snapshot>)でもよい。

b) RETRIEVE文

<retrieve-statement>::=RETRIEVE〔INTO <rel-name>〕
(<target-list>)
〔WHERE <qual>〕;

RETRIEVE文は、<qual> と <target-list> を満足する結果リレーション(<rel-name>)を生成する。<rel-name>が指定されていないときは、temporary file に生成する。

RETRIEVE文で導出された結果リレーションは、このセッションが終了した時点で消える。従って、再び使用するRETRIEVE文を退避するにはSAVEコマンドを用いる。退避されたパーマネントファイルから使用する場合はLOADコマンドを用いて、ローディングする。

c) DEFINE文

<define-statement>::=DEFINE <rel-name> (<target-list>)
〔WHERE <qual>〕 ;

DEFINE文は、LCSリレーション and/or viewリレーション上に、新たにviewリレーション(<rel-name>)を定義するものである。

view (DEFINE文)は、セッション終了後も、パーマネントファイル(VIEW-SAVEリレーション)に退避されるので、DESTROYコマンドで消去するまで保存される。

d) EDITコマンド

<edit-command>::=EDIT <line-no>〔<statement>〕;

EDITコマンドは行単位でのupdateを行なう。

1) EDIT <line-no> <statement> ; のとき

<line-no>がQBFR内にあれば、そのstatementを入力された。

<statement>で置き換える。QBFR内に無いときは、<line-no>に従って

<statement> を挿入する。

2) EDIT <line-no> ; のとき

QBFR内の<line-no> に対応するstatementを削除する。削除した後は、以降のstatementを前につめる。

c) GOコマンド

<go-command>::=GO 〔<line-no> | <rrel-name>〕;

1) GO; のとき

最新のQUEL実行文(retrieve, (append, delete, replace))を実行する。

2) GO <line-no>;

QBFR内の<line-no>から格納されているQUEL実行文を実行する。

3) GO <rrel-name>;

<rrel-name>を生成するretrieve文を実行する。

RRTBLにあれば、既にtree構造となっている問合せの再実行である。

f) DISPLAYコマンド

<display-command>::=DISPLAY〔<line-no> 〔, <line-no>〕〕;

QBFRの内容を表示させるコマンドである。

1) DISPLAY;のとき

QBFRの内容を順に全て出力する。

2) DISPLAY <line-no>;のとき

QBFR内のline-noを指定して、その組を出力する。

3) DISPLAY $<line\text{-}no>_1$, $<line\text{-}no>_2$; のとき

QBFR内の$<line\text{-}no>_1$ から$<line\text{-}no>_2$ までを出力する。

g) RENUMBERコマンド

<renumber-command>::= RENUMB R;

QBFR内のline-noを10から10単位にrenumberingする。

h) STATUSコマンド

<status-command>::= STATUS〔<line-no> | <rrel-name>〕;

1) STATUS;

最新のQUEL実行文(retrieve, (append, delete, replace))の実行状態を表示する。

2) STATUS <line-no>;

QBFR内の<line-no>から格納されているQUEL実行文の実行状態を表示する。

3) STATUS <rrel-name> ;

<rrel-name>を生成するQUEL実行文の実行状態を表示する。

i) OUTPUTコマンド

<output-command>::=OUTPUT〔<line-no> | <rrel-name>〕;

1) OUTPUT;

最新のQUEL実行文(retrieve, (append, delete, replace))の実行結果を端末に出力する。

2) OUTPUT <line-no>;

QBFR内の<line-no>から格納されているQUEL実行文の実行結果を端末に出力する。

3) OUTPUT <rrel-name>;

<rrel-name>を生成するQUEL実行文の実行結果を端末に出力する。

j) CANCELコマンド

<carcel-command>::=CANCEL〔<line-no> | <rrel-name>〕;

1) CANCEL;

最新のQUEL実行文(retriene, (append, delete, replace))で生成されたCOBOL DML プログラムの実行をキャンセルする。

2) CANCEL <line-no>;

QBFR内<line-no>から格納されているQUEL実行文で生成されたCOBOL DML プログラムの実行をキャンセルする。

3) CANCEL <rrel-name>;

<rrel-name>を生成するQUEL実行文で生成されたCOBOL DML プログラムの実行をキャンセルする。

k) SAVEコマンド

<save-command>::=SAVE <rrel-name> <datasct-name>
VOL(<volume-name>);

<rrel-name>に対する実行結果を、<volume-name>のボリュームに<dataset-name>のファイルをアロケートし、内容をコピーする。

1.5 IPFによるRISCOSとAIMのインタフェース

RISCOSでは、QUEL問合せからCOBOL DMLプログラムを生成しており、このプログラムを生成しており、このプログラムを実行し、ユーザに出力することが必要である。LDP-V1.5では、Jipnetの対話型プロトコルITPを利用したローカル通信機能により、COBOL DMLプログラムを実行させ、ユーザに出力するシステムを実現した。しかし、ローカル通信機能には、オーバヘッドの負担と通信の間端末を占有するなどの欠点を持っている。

そこで、RISCOSとAIMとのインタフェースとしてメーカで提供するIPF(会話型プロ

グラミング機能）を利用し、ＲＩＳＣＯＳで生成したＣＯＢＯＬ　ＤＭＬプログラムを実行させ、ユーザに出力する方法を検討する。

ＩＰＦでは、ＰＬ／Ｉ（あるいはＣＯＢＯＬ，ＦＯＲＴＲＡＮ）プログラム内からＴＳＳコマンドを実行させるＩＰＦＣＭＤサブルーチンを提供している。ＩＰＦＣＭＤは、使用するＴＳＳコマンドをメッセージとして受け取り実行するサブルーチンである。

ＲＩＳＣＯＳでは、このＩＰＦＣＭＤを使用して、ＲＩＳＣＯＳとＡＩＭのインタフェースを構築する。ＩＰＦＣＭＤを利用して実現するインタフェースの機能としては、次のものがある。

・生成されたＣＯＢＯＬ　ＤＭＬ　プログラムの実行（ＧＯコマンド）

・ＣＯＢＯＬ　ＤＭＬ　プログラムの実行結果の出力（ＯＵＴＰＵＴコマンド）

・ジョブの状態表示　（ＳＴＡＴＵＳコマンド）

・ジョブのキャンセル　（ＣＡＮＣＥＬコマンド）

・実行結果のセーブ　（ＳＡＶＥコマンド）

Ａ　ＩＰＦインタフェース機能の管理

ＩＰＦによって実現される機能を使用する上で、次の様な制限がある。

・検索結果の端末の出力を行なう為には、ジョブ名としてユーザＩｄ＋１文字でなければならない。

・同時に複数のジョブを実行させるためには、共有出来るリソース以外は、各ジョブ毎に確保する必要がある（ＣＯＢＯＬ　ＤＭＬ出力ファイル及び検索結果ファイルｅｔｃ）。

この為、ＲＩＳＣＯＳでは、各ユーザＩｄ毎に同時に実行させるジョブ数を決め、各ジョブ毎にＣＯＢＯＬ　ＤＭＬ出力ファイル及び検索結果ファイルをあらかじめＵＳＲＴＢＬ（表１）に割り当てておく。

表１　ＵＳＲＴＢＬ

属　性　名	属性番号	ｒｏｌｅ	ｔｙｐｅ	ｌｅｎｇｔｈ	説　　明
Ｊｏｂ－ｎａｍｅ	１	Ｋ	Ｃ	８	ＴＳＳユーザＩｄ＋１文字
ＤＭＬ ｄａｔａｓｅｔ－ｎａｍｅ	２	Ｎ	Ｃ	１６	ＣＯＢＯＬ　ＤＭＬ　プログラム出力用データセット名
ＲＥＳＵＬＴ ｄａｔａｓｅｔ－ｎａｍｅ	３	Ｎ	Ｃ	１６	実行結果が出力されるデータセット名
ｂｕｓｙ－ｆｌａｇ	４	Ｎ	Ｄ	１	ＤＭＬ　プログラム＆結果出力用データセットが使用中（＝１）であることを示す

ＱＵＥＬ問合せの実行コマンド（ＧＯコマンド）が入力された時、このＵＳＲＴＢＬをサーチし、使用されていないジョブ名、ＤＭＬ出力データセット名及び実行結果出力データセット名をこの問合せに割り当てる。使用されていないジョブ名が無い場合、使用している問合せの結果リレーション名を表示し、不用な問合せに対してジョブのキャンセルを行なう。

使用されていないものが見つかった場合は、ＵＳＲＴＢＬからジョブ名、ＤＭＬ出力データセット名及び実行結果データセット名をＲＲＴＢＬ（表２）の該当欄に転送する。それ以後のジョブ結果の出力、ジョブの状態表示等はＲＲＴＢＬによって管理を行う。

ジョブ結果の出力、ジョブのキャンセル及び結果のセーブを実行すると、その問合せが使用していたりソース（ジョブ名、ＤＭＬ出力データセット、実行結果データセット）を解放する。即ち、ＲＲＴＢＬの該当欄のクリアーとＵＳＲＴＢＬのｂｕｓｙ－ｆｌａｇをゼロ（未使用）に設定する。

これらのコマンドを管理するために、結果リレーションの情報を保持しているＲＲＴＢＬを使用する。

表２　ＲＲＴＢＬ

属　性　名	属性番号	role	type	length	説　　明
rrel-name	1	K	C	16	result relation名又は view name
degree	2	N	D	2	result relation の attribute 数
type	3	N	C	2	V＝define　D＝delete R＝retrieve　RP＝replace A＝append
head-adr	4	N	C	4	rrel-nameの結果がstoreされているSWFのhead address（byte address）
tail-adr	5	N	C	4	rrel-nameの結果がstoreされているSWFのtail-address（byte address）
qpt	6	N	P	2	この問合せのＱ－treeへの pointer
ジョブ名	7	N	C	8	生成されたＣＯＢＯＬ　ＤＭＬプログラムを実行させるジョブ名
ＤＭＬ　ＤＡＴＡＳＥＴ名	8	N	C	16	生成されたＣＯＢＯＬ　ＤＭＬプログラムが出力されるデータセット名
結果データセット名	9	N	C	16	実行結果が出力されるデータセット名
busy flag	10	N	D	1	ＤＭＬ　ＤＡＴＡＳＥＴ及び結果データセットが使用中（＝１）であることを表す。

例としては、ユーザＩＤ＝ＬＤＰ０１１０．同時実行ジョブ数＝５とする。

ＤＭＬデータセット名＝ＣＯＢＯＬＦ１～５、結果データセット名＝ＲＥＳＵＬＴ１～５とする。この時、ＵＳＲＴＢＬは次の様に設定する。

ジョブ名	ＤＭＬデータセット	結果データセット	busy-flag
ＬＤＰ０１１０１	ＣＯＢＯＬＦ１	ＲＥＳＵＬＴ１	0
ＬＤＰ０１１０２	ＣＯＢＯＬＦ２	ＲＥＳＵＬＴ２	0
ＬＤＰ０１１０３	ＣＯＢＯＬＦ３	ＲＥＳＵＬＴ３	0
ＬＤＰ０１１０４	ＣＯＢＯＬＦ４	ＲＥＳＵＬＴ４	0
ＬＤＰ０１１０５	ＣＯＢＯＬＦ５	ＲＥＳＵＬＴ５	0

問合せに対してＧＯコマンドが入力された時、このＵＳＲＴＢＬをサーチして、ｂｕｓｙ－ｆｌａｇがＯＦＦのジョブ名、ＤＭＬデータセット名及び結果データセット名を割り当てる（ＲＲＴＢＬの該当欄に転送し、ｂｕｓｙ－ｆｌａｇをＯＮにする）。

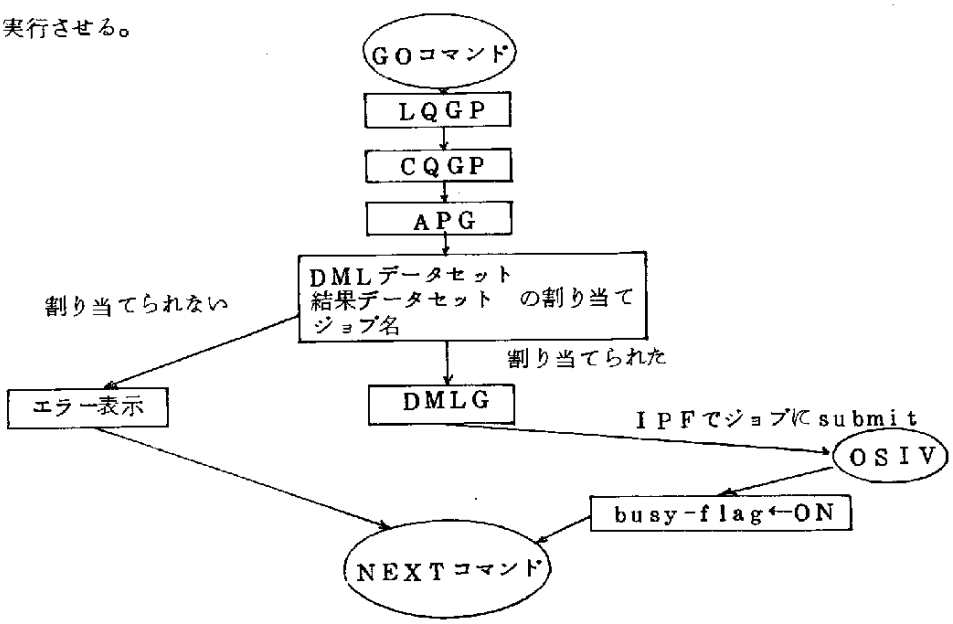
Ｂ　ＩＰＦインターフェースのコマンド

ＩＰＦを使用したインタフェースの各コマンドの処理について述べる。

ａ）ＧＯコマンド

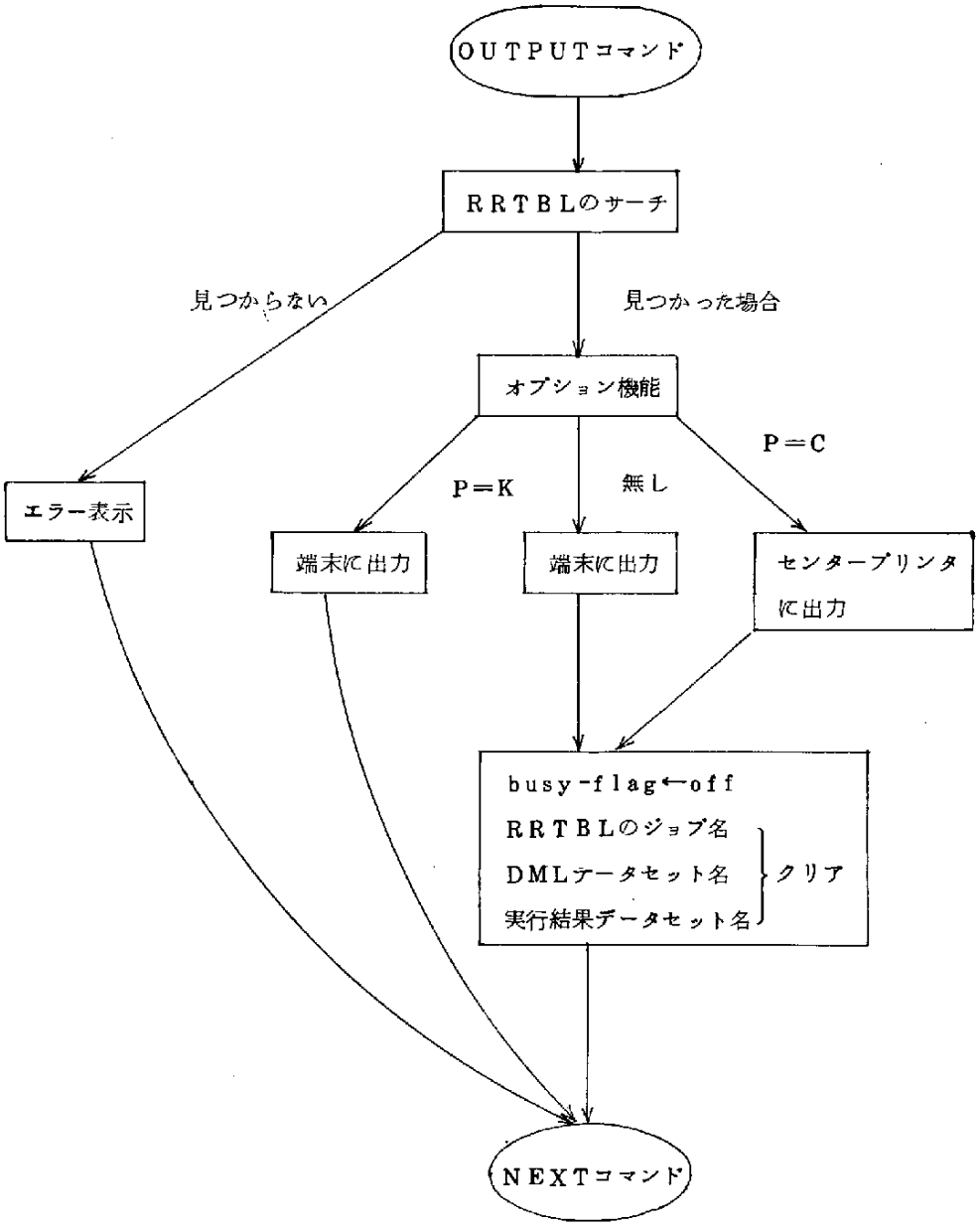
ＧＯコマンドは、ＱＵＥＬ問合せを入力し、問合せ変換でＣＯＢＯＬ　ＤＭＬ　プログラムを生成し、これをＡＩＭ上で実行させるコマンドである。

ＧＯコマンドが入力されると、問合せ変換で生成されるＣＯＢＯＬ　ＤＭＬ　プログラムに対するデータセット、実行結果の為のデータセット及びジョブ名の割り当てを行なう。割り当てることが出来たら、ＣＯＢＯＬ　ＤＭＬ　プログラムを生成し、ＩＰＦを使用してジョブを実行させる。



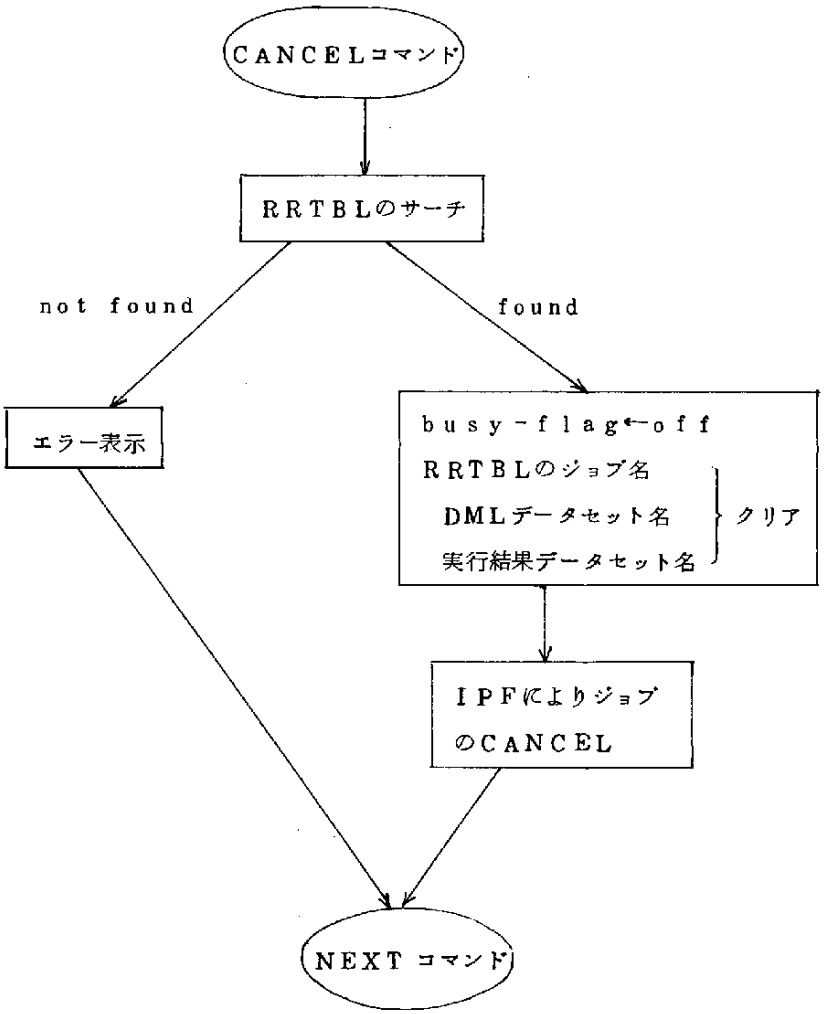
b) ＯＵＴＰＵＴコマンド

ＯＵＴＰＵＴコマンドは、実行結果を出力する為のコマンドである。実行結果は、OUTPUTコマンドにより端末へ出力される。オプションとして、実行結果を端末へ出力した後も保存する機能（Ｐ＝Ｋ（ｅｅｐ））と、センタープリンターへの出力させる機能（Ｐ＝Ｃ（ｅｎｔｅｒ））を持つ。



c) CANCEL コマンド

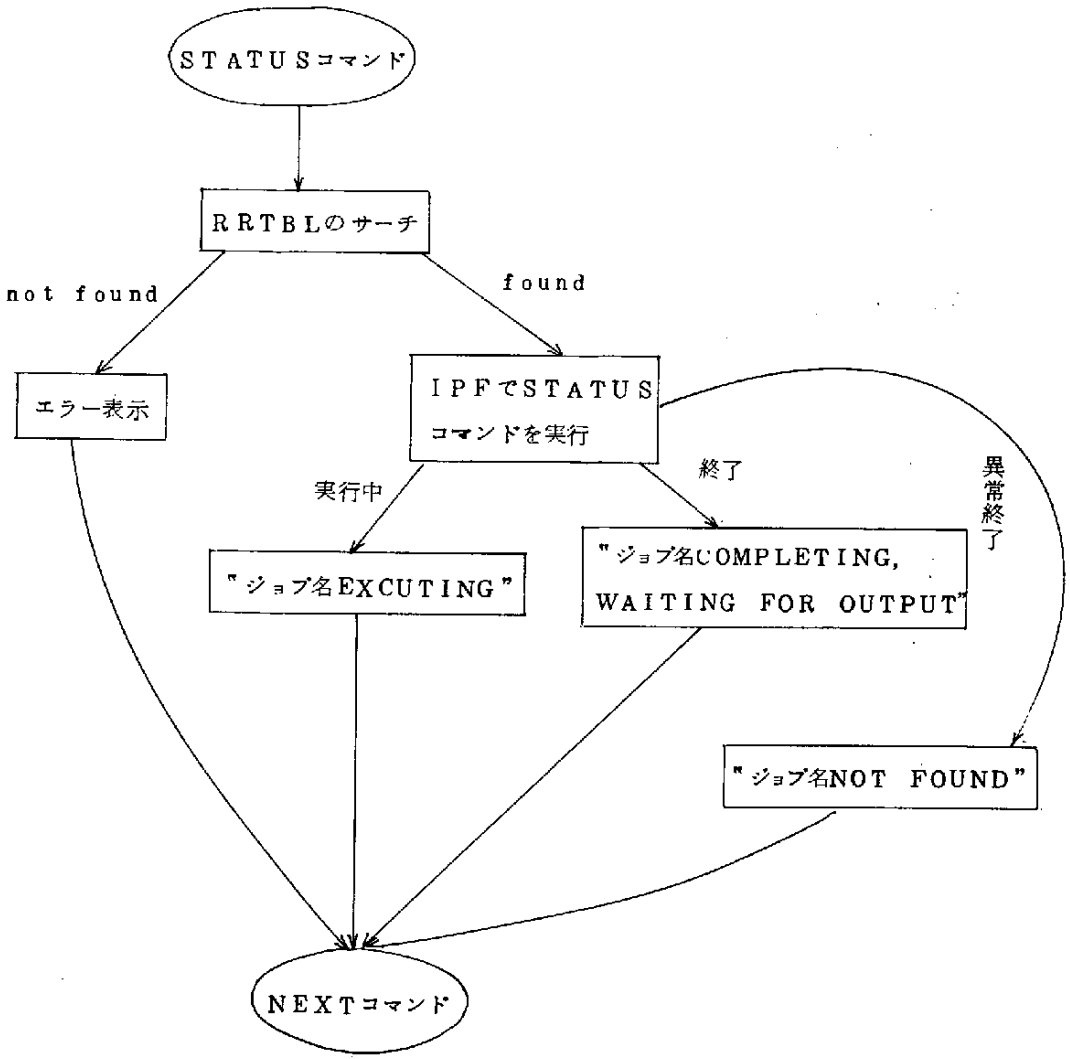
CANCELコマンドは、実行された問合せの実行をキャンセルする為のコマンドである。CANCELコマンドにより、キャンセルされた問合せに割り当てられていたジョブ名、DMLデータセット名及び実行結果データセットの割り当てを解放する。



d) STATUSコマンド

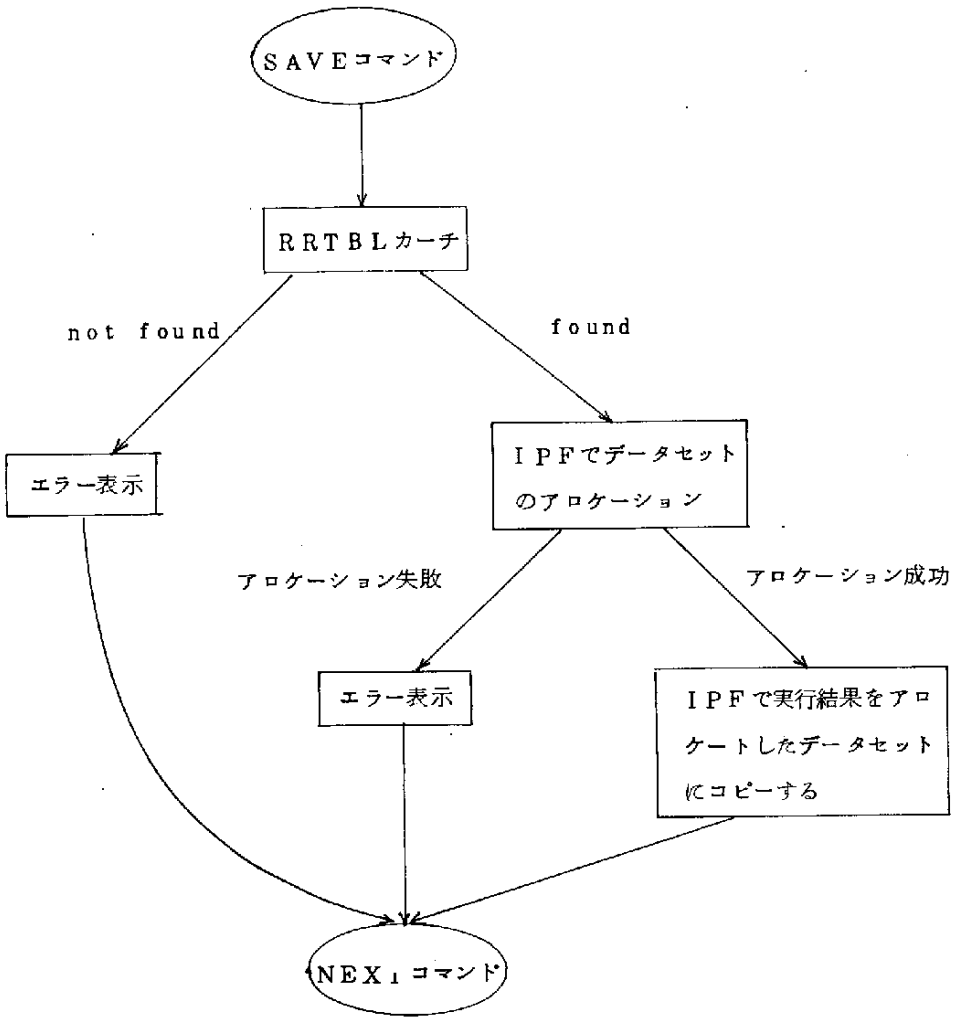
STATUSコマンドは、実行された問合せが終了したかどうかを見るためのコマンドである。"ジョブ名EXCUTING"が表示された場合は、またジョブが実行中であるかを示す。また、"ジョブ名COMPLETING, WAITING FOR OUTPUT"に表示された場合は、ジョブが終了していることを示す。

COBOL DML プログラムの実行以前に異常終了したジョブに対しては、"ジョブ名 NOT FOUND"が表示される。



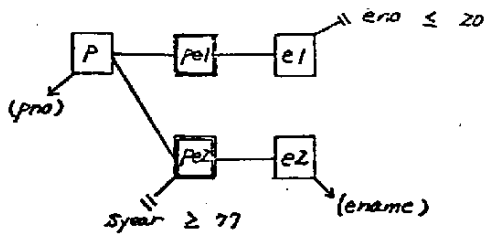
c) SAVEコマンド

SAVEコマンドは、<rrel-rame>に対する実行結果をパーマネントファイルとしてセーブする為のコマンドである。SAVEコマンドでは、必らずボリューム名及びデータセット名を指定する。


SAVEコマンド
RRTBLカーチ
not found
found
エラー表示
IPFでデータセット
のアロケーション
アロケーション失敗
アロケーション成功
エラー表示
IPFで実行結果をアロ
ケートしたデータセット
にコピーする
NEX1コマンド

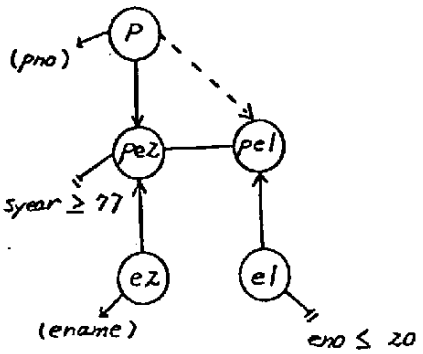
付記 2　　評価用問合せとアクセス木

DFA 010

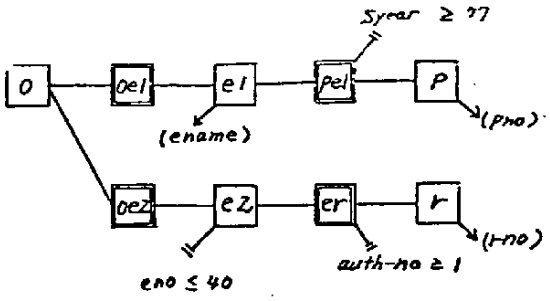



```
**DFA010
RANGE ( P, PROJECT ) ( E1, EMPLOYEE ) ( E2, EMPLOYEE );
RANGE ( PE1, PROJ-EMP-LNK ) ( PE2, PROJ-EMP-LNK );
RETRIEVE  INTO  DFA010  ( P.PNO, E2.ENAME )
WHERE
      P.#P  =  PE1.#P    AND    PE1.#E  =  E1.#E   AND
      P.#P  =  PE2.#P    AND    PE2.#E  =  E2.#E   AND
      PE2.SYEAR GE  77   AND    E1.ENO  LE  20;
```

DFA 1

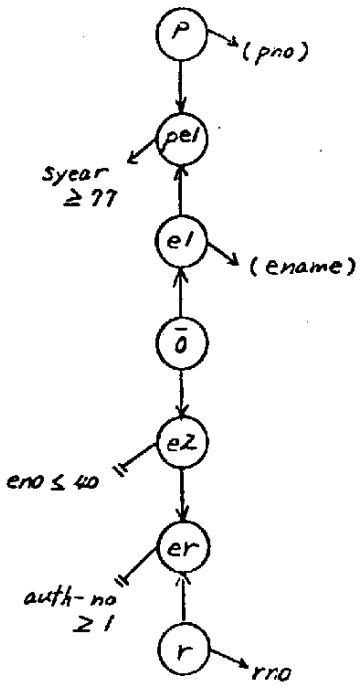


DFA 020

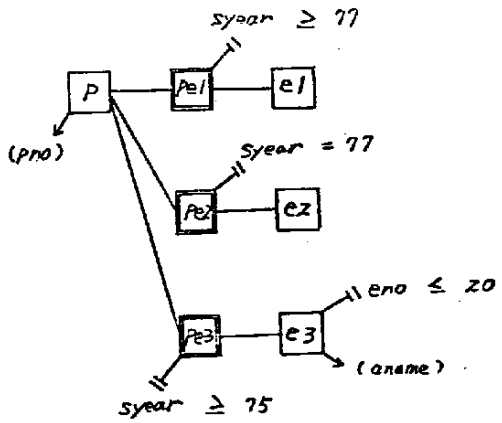



```
**DFA020
RANGE ( ER, EMP-REP-LNK );
RANGE ( O, ORGANIZATION ) ( R, REPORTR );
RANGE ( OE1, ORG-EMP ) ( OE2, ORG-EMP );
RANGE ( P, PROJECT ) ( E1, EMPLOYEE ) ( E2, EMPLOYEE );
RANGE ( PE1, PROJ-EMP-LNK ) ( PE2, PROJ-EMP-LNK );
RETRIEVE  INTO  DFA020  ( E1.ENAME, P.PNO, R.RNO )
WHERE
      O.@ORGANIZATION  =  OE1.@ORGANIZATION        AND
      O.@ORGANIZATION  =  OE2.@ORGANIZATION        AND
      OE1.@E  =  E1.@E    AND   E1.@E  =  PE1.@E   AND
      PE1.@P  =  P.@P     AND
      OE2.@E  =  E2.@E    AND   E2.@E  =  ER.@E    AND
      ER.@R   =  R.@R     AND   E2.ENO LE 40       AND

      PE1.SYEAR  GE 77    AND   ER.AUTH-NO  GE   1;
```

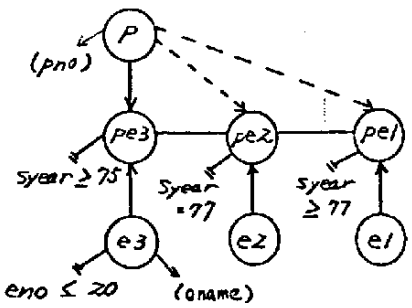


DFA 030

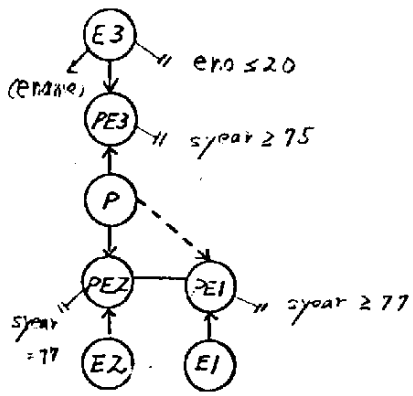



```
**DFA030
RANGE ( P, PROJECT ) ( E1, EMPLOYEE );
RANGE ( E2, EMPLOYEE ) ( E3, EMPLOYEE );
RANGE ( PE1, PROJ-EMP-LNK ) ( PE2, PROJ-EMP-LNK ) ( PE3, PROJ-EMP-LNK );
RETRIEVE  INTO   DFA030      ( P.PNO, E3.ENAME )
WHERE
      P.#P   =  PE1.#P   AND   P.#P   =  PE2.#P   AND
      P.#P   =  PE3.#P   AND   PE1.#E =  E1.#E    AND
      PE2.#E =  E2.#E    AND   PE3.#E =  E3.#E    AND
      PE1.SYEAR  GE 77   AND   PE2.SYEAR  =  77   AND
      PE3.SYEAR  GE 75   AND   E3.ENO     LE  20;
```

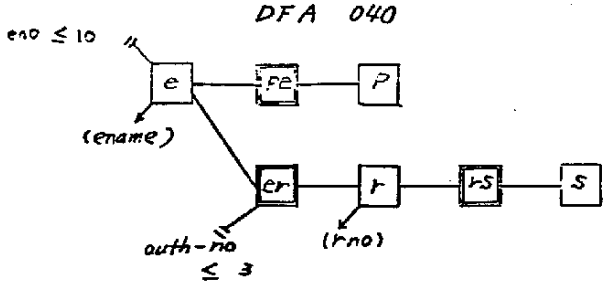
DFA 1



DFA 2, DFA 3

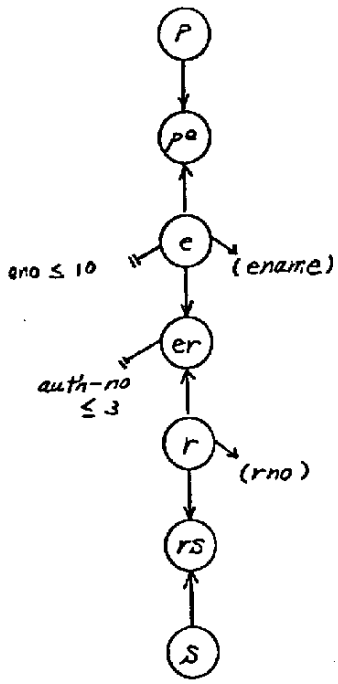


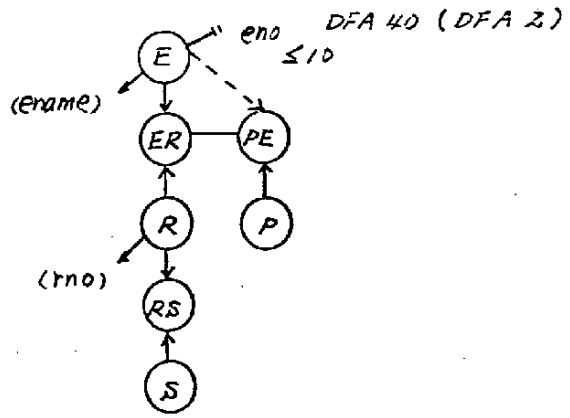
DFA 040

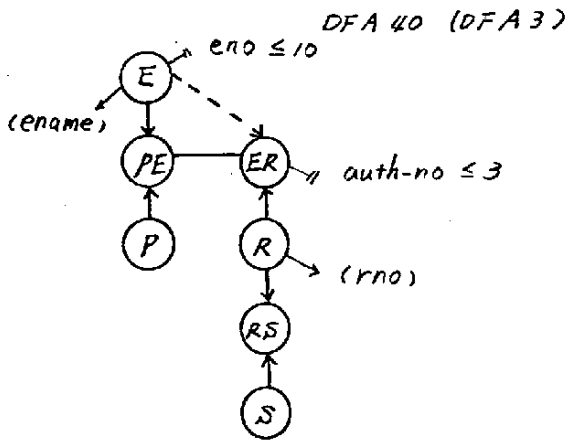



```
**DFA040
RANGE ( E, EMPLOYEE ) ( S, SUBJECT ) ( R, REPORTR );
RANGE ( P, PROJECT ) ( EP, EMP-REP-LNK );
RANGE ( PE, PROJ-EMP-LNK ) ( RS, REP-SUBJ );
RETRIEVE  INTO   DFA040   ( E.ENAME, R.RNO )
WHERE
        E.ENO  LE  10  AND   E.EE  =  PE.EE          AND
        PE.EP  = P.EP  AND   E.EE  =  ER.EE          AND
        E.ER   = R.ER  AND   R.SP  =  RS.SP          AND
        RS.SJ  = S.SJ  AND
        ER.AUTH-NO  LE  3;
```

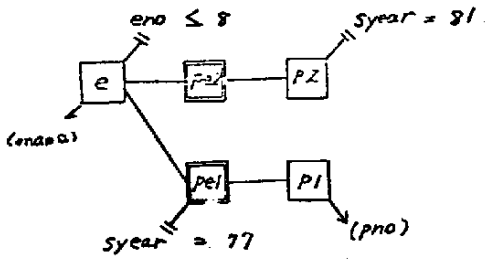
DFA 1




DFA 40 (DFA 2)
eno
≤10
E
(ename)
ER
PE
R
P
(rno)
RS
S

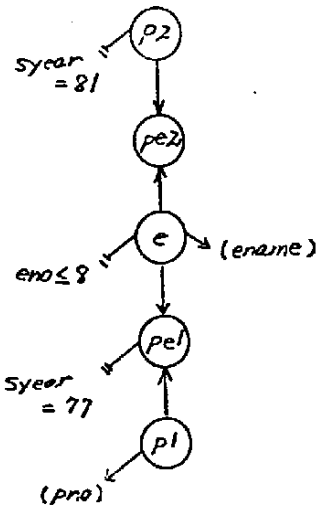

DFA 40 (DFA 3)
eno ≤10
E
(ename)
PE
ER
auth-no ≤3
P
R
(rno)
RS
S

DFA 050

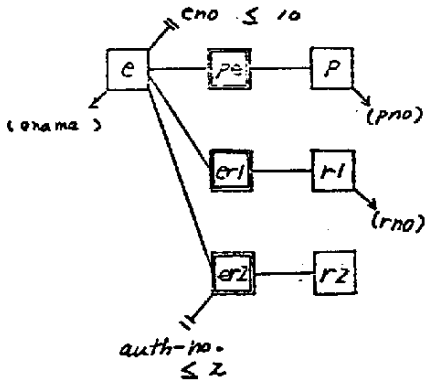



```
**DFA050
RANGE ( E, EMPLOYEE ) ( P1, PROJECT ) ( P2, PROJECT );
RANGE ( PE1, PROJ-EMP-LNK ) ( PE2, PROJ-EMP-LNK );
RETRIEVE  INTO   DFA050      ( E.ENAME, P1.PNO )
WHERE
      P2.SYEAR  =  81        AND
      P2.@P     =  PE2.@P    AND   PE2.@E   =  E.@E    AND
      E.@E      =  PE1.@E    AND   PE1.@P   =  P1.@P   AND
      PE1.SYEAR =  77        AND   E.ENO    LE  8;
```

DFA 1, DFA 2, DFA 3



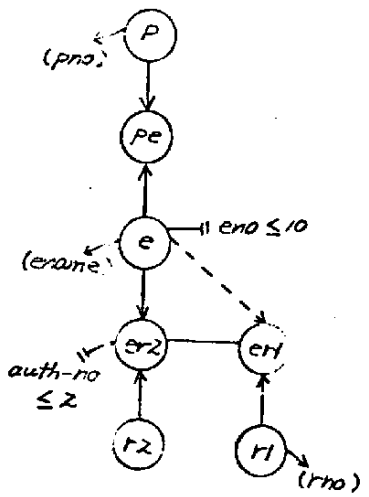
DFA 060



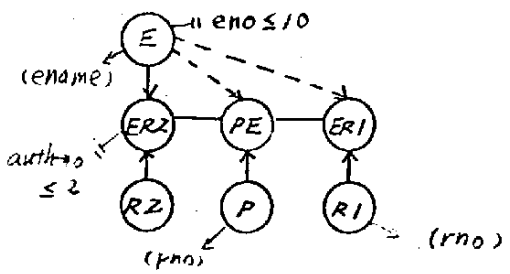

```
**DFA060
RANGE ( E, EMPLOYEE ) ( P, PROJECT );
RANGE ( R1, REPORTR ) ( R2, REPORTR );
RANGE ( ER1, EMP-REP-LNK ) ( ER2, EMP-REP-LNK );
RANGE ( PE,  PROJ-EMP-LNK );

RETRIEVE  INTO   DFA060     ( E.ENAME, R1.RNO, P.PNO )
WHERE
      E.@E  =  PE.@E   AND   PE.@P  =  P.@P    AND
      E.@E  =  ER1.@E  AND   ER1.@R =  R1.@R   AND
      E.@E  =  ER2.@E  AND   ER2.@R =  R2.@R   AND
      E.ENO  LE  10    AND   ER2.AUTH-NO  LE  2;
```

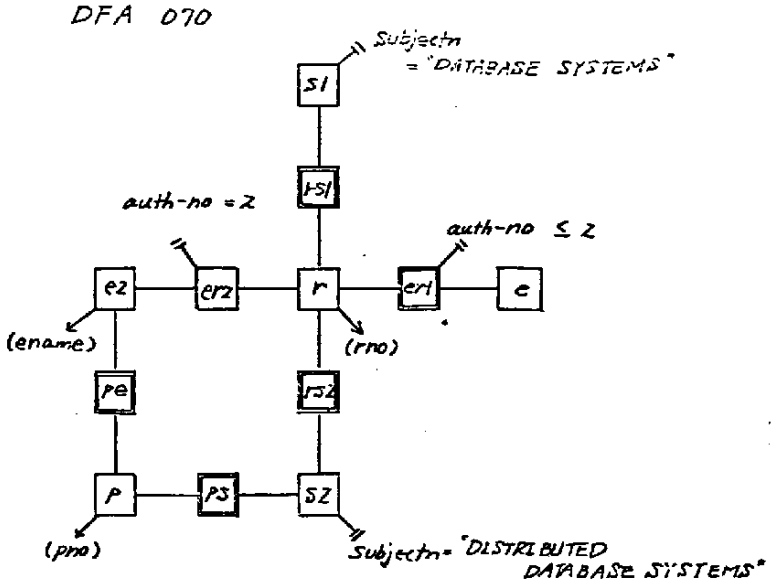
DFA 1



DFA 2, DFA 3

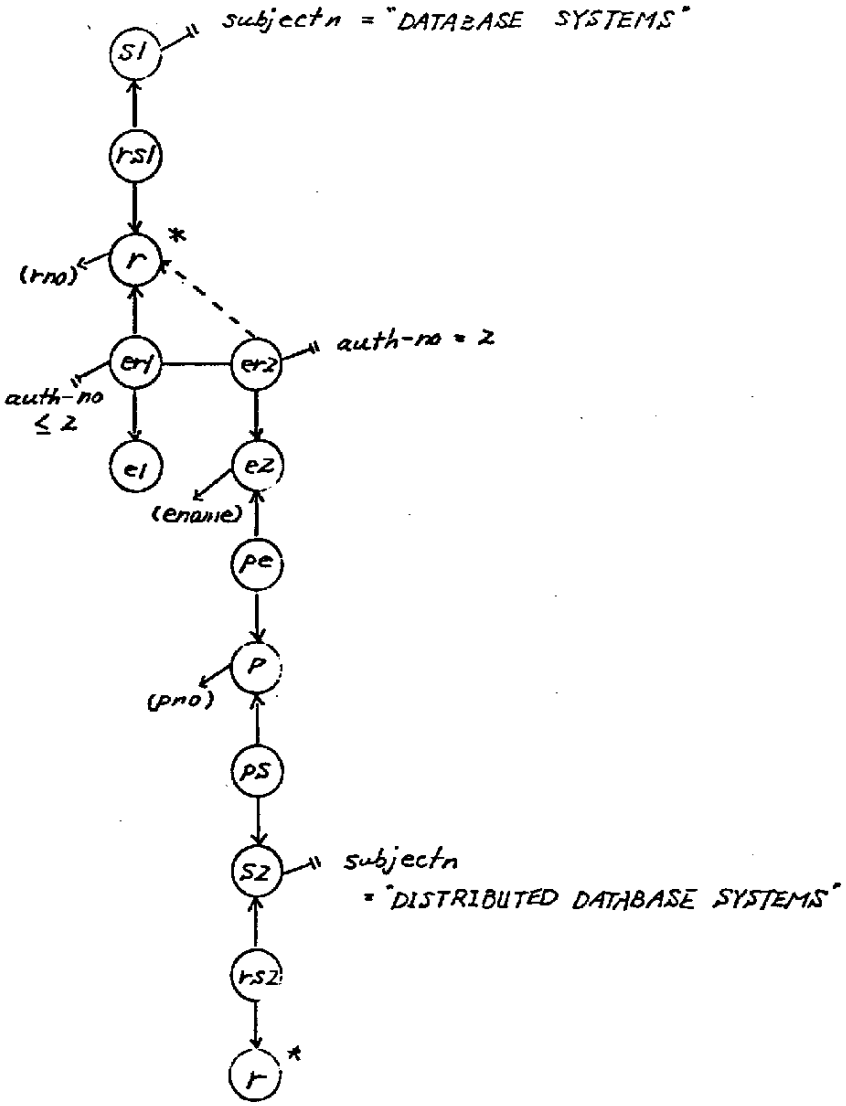


DFA 070

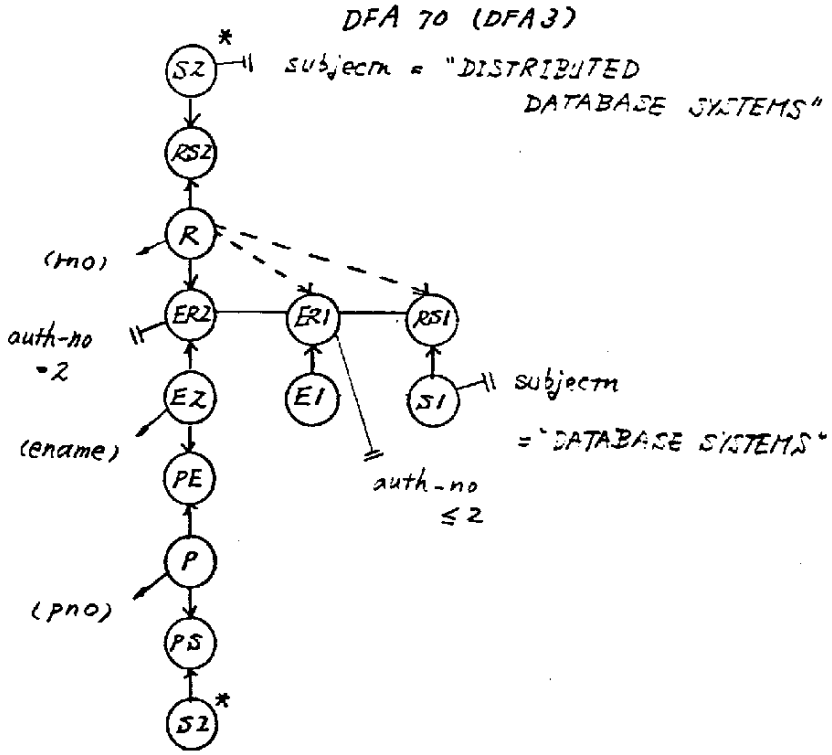



```
**DFA070
RANGE ( E1, EMPLOYEE ) ( E2, EMPLOYEE );
RANGE ( R, REPORTR ) ( P, PROJECT );
RANGE ( S1, SUBJECT ) ( S2, SUBJECT );
RANGE ( ER1, EMP-REP-LNK ) ( ER2, EMP-REP-LNK );
RANGE ( RS1, REP-SUBJ ) ( RS2, REP-SUBJ );
RANGE ( PS, PROJ-SUBJ ) ( PE, PROJ-EMP-LNK );
RETRIEVE  INTO  DFA070    ( E2.ENAME, P.PNO, R.RNO )
WHERE
      S1.SUBJECTN  =  "DATABASE SYSTEMS"          AND
      S1.@J  =  RS1.@J                            AND
      RS1.@R  =  R.@R   AND   R.@R   =  RS2.@R    AND
      RS2.@J  =  S2.@J                            AND
      S2.SUBJECTN  = "DISTRIBUTED DATABASE SYSTEMS"    AND
      S2.@J  =  PS.@J                             AND
      PS.@P  =  P.@P    AND  P.@P   =  PE.@P      AND
      PE.@E  =  E2.@E  AND  E2.@E  =  ER2.@E      AND
      R.@R   =  ER1.@R AND  ER1.@E =  E1.@E       AND
      ER2.@R =  R.@R    AND
      ER1.AUTH-NO  LE  2      AND   ER2.AUTH-NO  =  2;
```

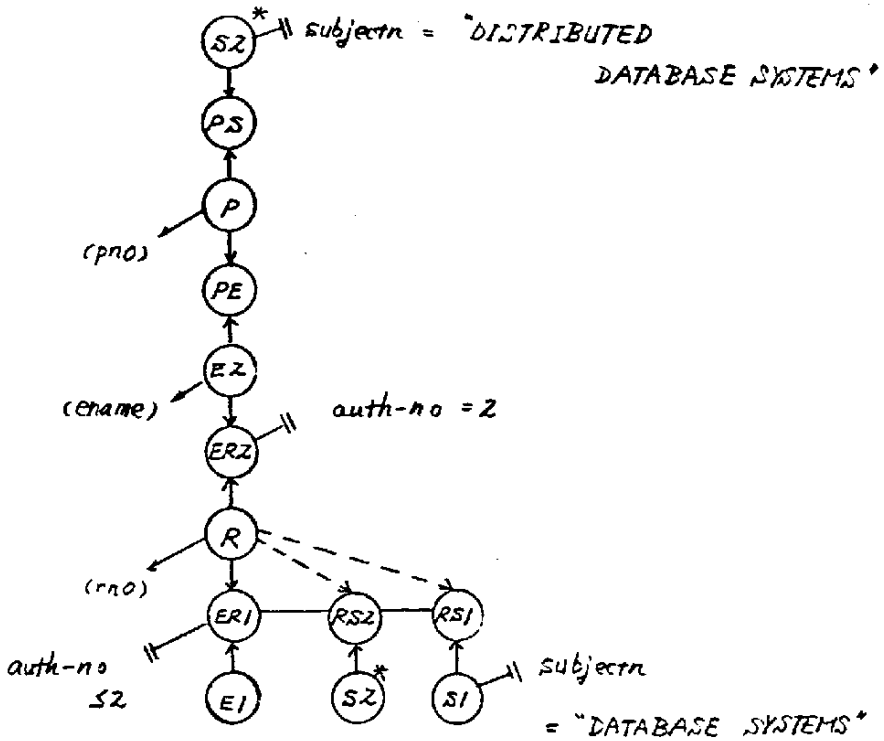
DFA 70 (DFA 1)


subjectn = "DATABASE SYSTEMS"
s1
rs1
r
*
(rno)
er1
er2
auth-no = z
auth-no ≤ z
e1
e2
(ename)
pe
P
(pno)
ps
s2
subjectn = "DISTRIBUTED DATABASE SYSTEMS"
rs2
r
*

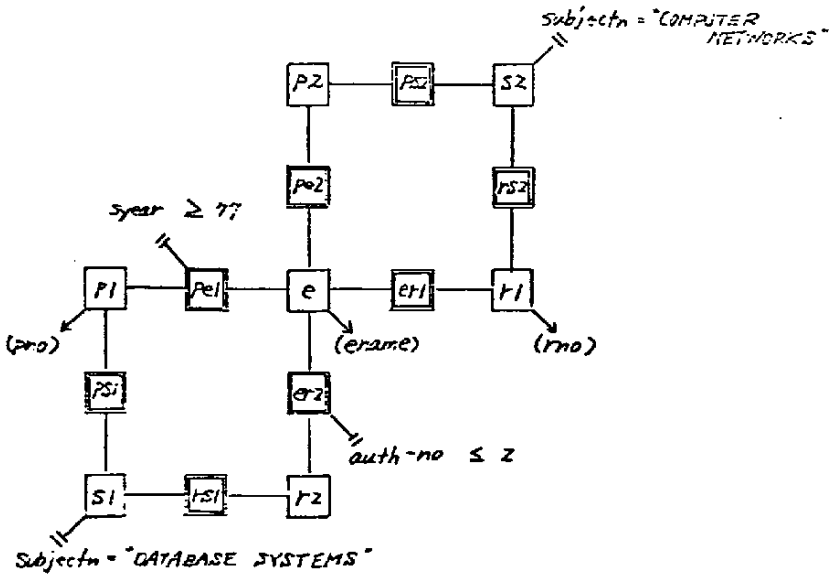
DFA 70 (DFA3)



DFA 70 (DFA2)

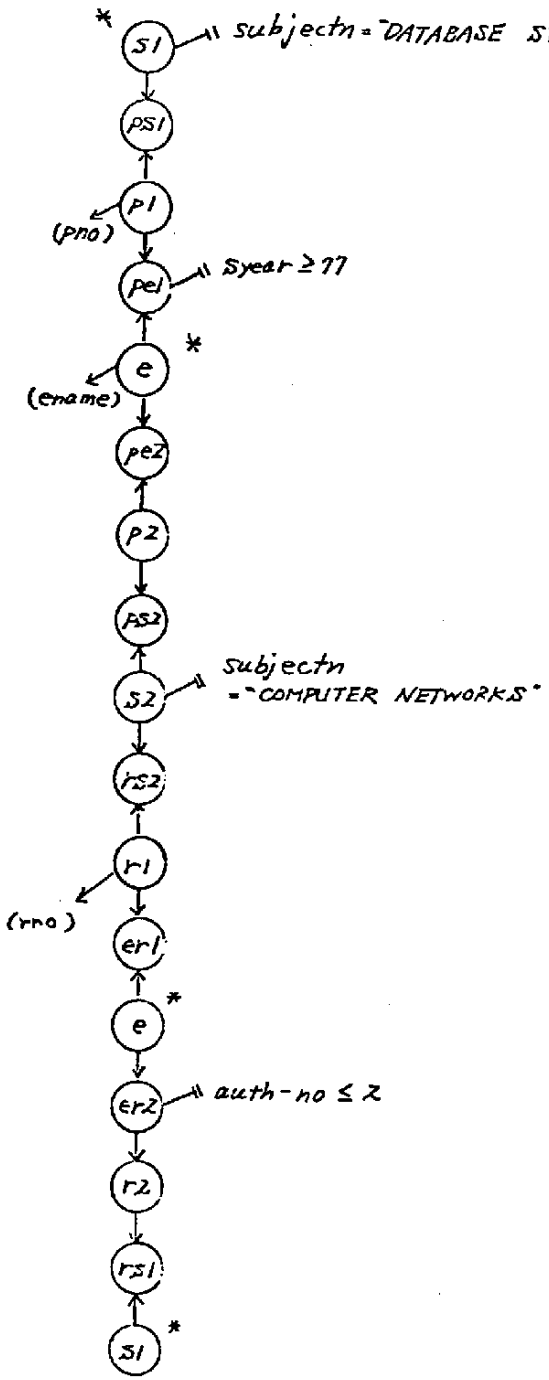


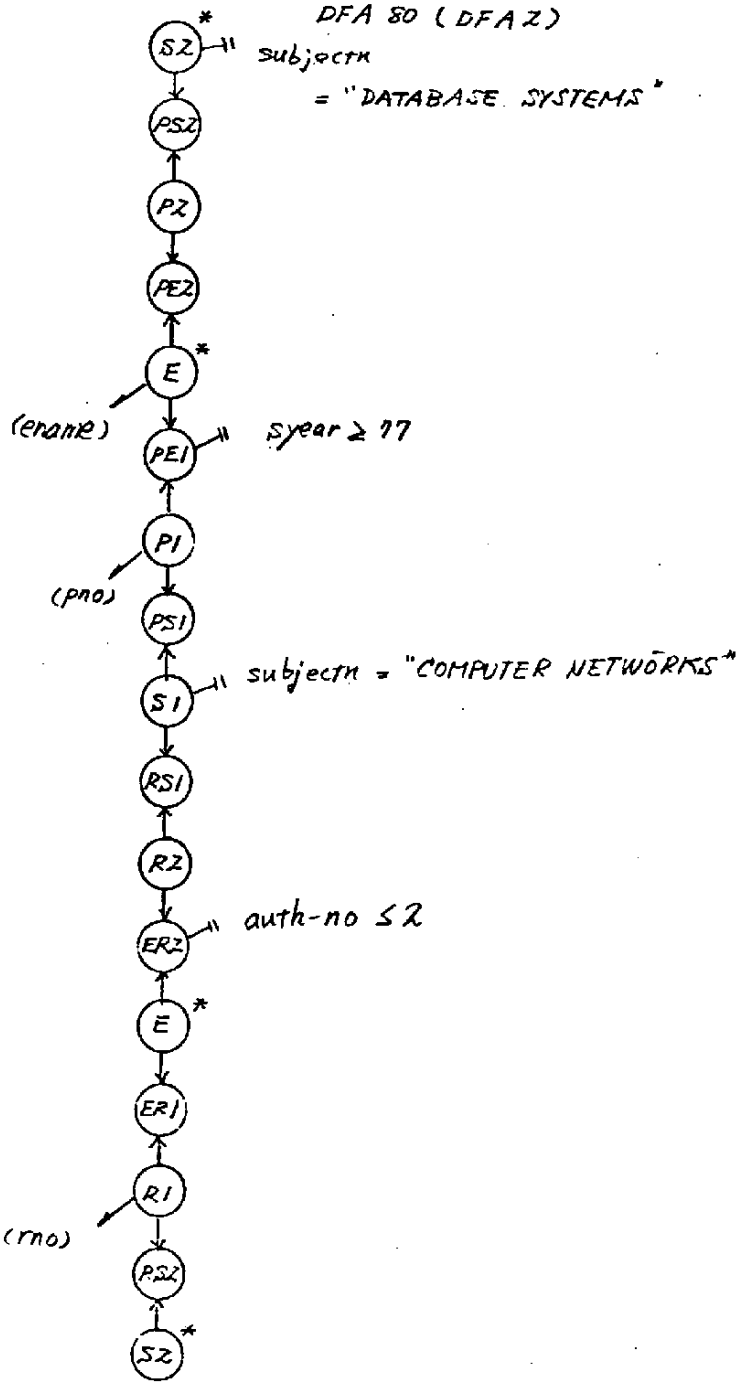
DFA 080



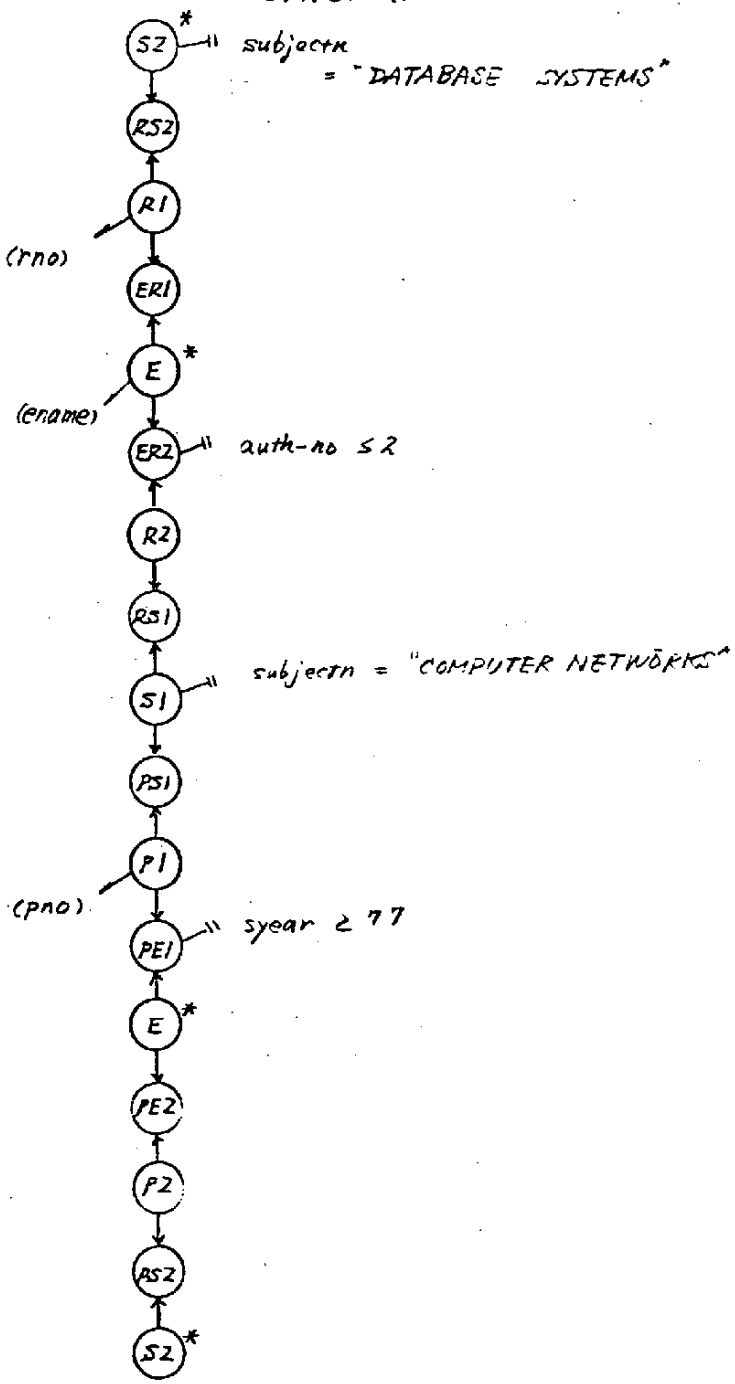

```
**DFA80
RANGE ( E, EMPLOYEE ) ( P1, PROJECT ) ( P2, PROJECT );
RANGE ( S1, SUBJECT ) ( S2, SUBJECT ) ( R1, REPORTR ) ( R2, REPORTR );
RANGE ( ER1, EMP-REP-LNK ) ( ER2, EMP-REP-LNK );
RANGE ( PE1, PROJ-EMP-LNK ) ( PE2, PROJ-EMP-LNK );
RANGE ( PS1, PROJ-SUBJ ) ( PS2, PROJ-SUBJ );
RANGE ( RS1, REP-SUBJ ) ( RS2, REP-SUBJ );
RETRIEVE  INTO   DFA080      ( P1.PNO, R1.RNO, E.ENAME )
WHERE
       S1.SUBJECTN  = "DATABASE SYSTEMS"                          AND
       S1.SJ  = RS1.SJ   AND  RS1.SR  =  R2.SR                    AND
       R2.SR  = ER2.SR   AND  ER2.SE  =  E.SE                     AND
       ER2.AUTH-NO  LE 2                                          AND
       E.SE   = PE1.SE   AND   PE1.SYEAR  GE  77                  AND
       PE1.SP = P1.SP    AND   P1.SP  =  PS1.SP                   AND
       PS1.SJ =  S1.SJ                                            AND
       E.SE   = ER1.SE   AND   ER1.SR =  R1.SR                    AND
       P1.SP  = PS2.SP                                            AND
       PS2.SJ =  S2.SJ                                            AND
       S2.SUBJECTN  = "COMPUTER NETWORKS"                         AND
       S2.SJ  =  PS2.SJ                                           AND
       PS2.SP =  P2.SP   AND   P2.SP  =  PE2.SP                   AND
       PE2.SE =  E.SE;
```

DFA 80 (DFA 1)

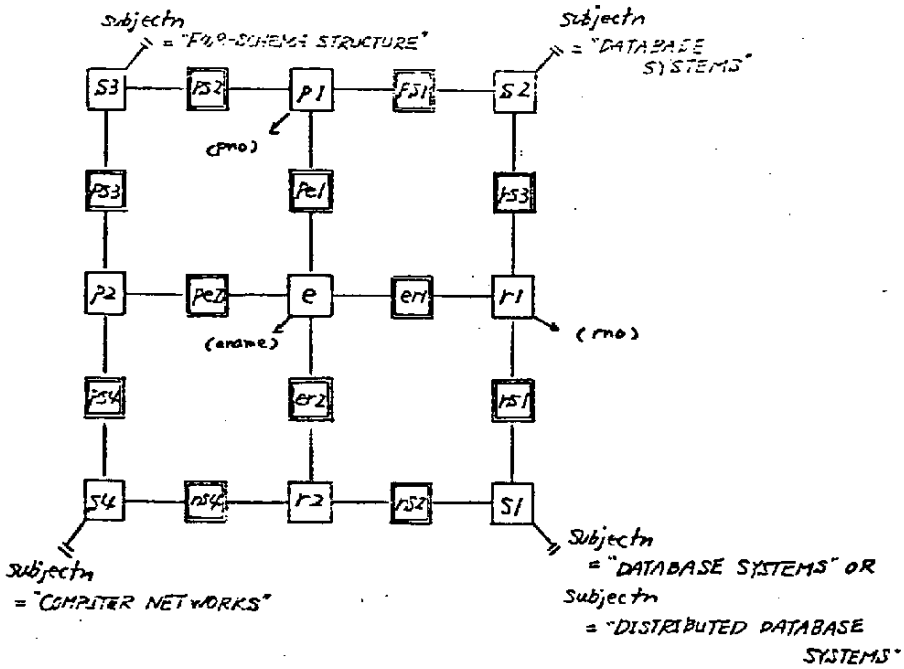

*
S1
subjectn = "DATABASE SYSTEMS"
PS1
P1
(pno)
Pe1
Syear ≥ 77
e
*
(ename)
Pe2
P2
PS2
S2
subjectn
= "COMPUTER NETWORKS"
rS2
r1
(rno)
er1
e
*
er2
auth-no ≤ 2
r2
rS1
S1
*


DFA 80 (DFA 2)
S2
subjectn
= "DATABASE SYSTEMS"
PS2
P2
PE2
E
(ename)
PE1
syear ≥ 77
P1
(pno)
PS1
S1
subjectn = "COMPUTER NETWORKS"
RS1
R2
ER2
auth-no ≤ 2
E
ER1
R1
(rno)
PS2
S2

DFA 80 (DFA 3)

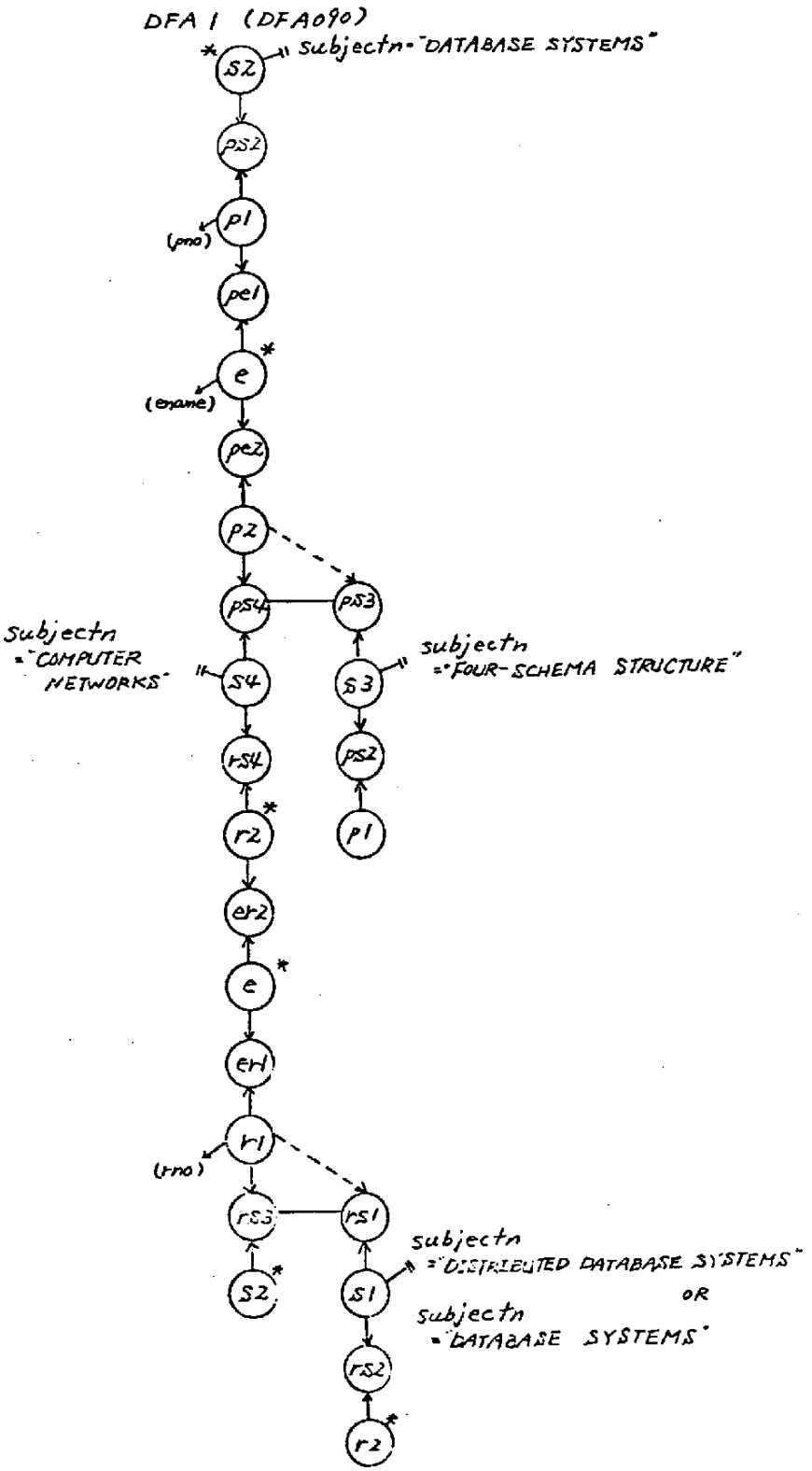

S2
subjectn = "DATABASE SYSTEMS"
RS2
R1
(rno)
ER1
E
(ename)
ER2
auth-no ≤ 2
R2
RS1
S1
subjectn = "COMPUTER NETWORKS"
PS1
P1
(pno)
PE1
syear ≥ 77
E
PE2
P2
PS2
S2

DFA 090

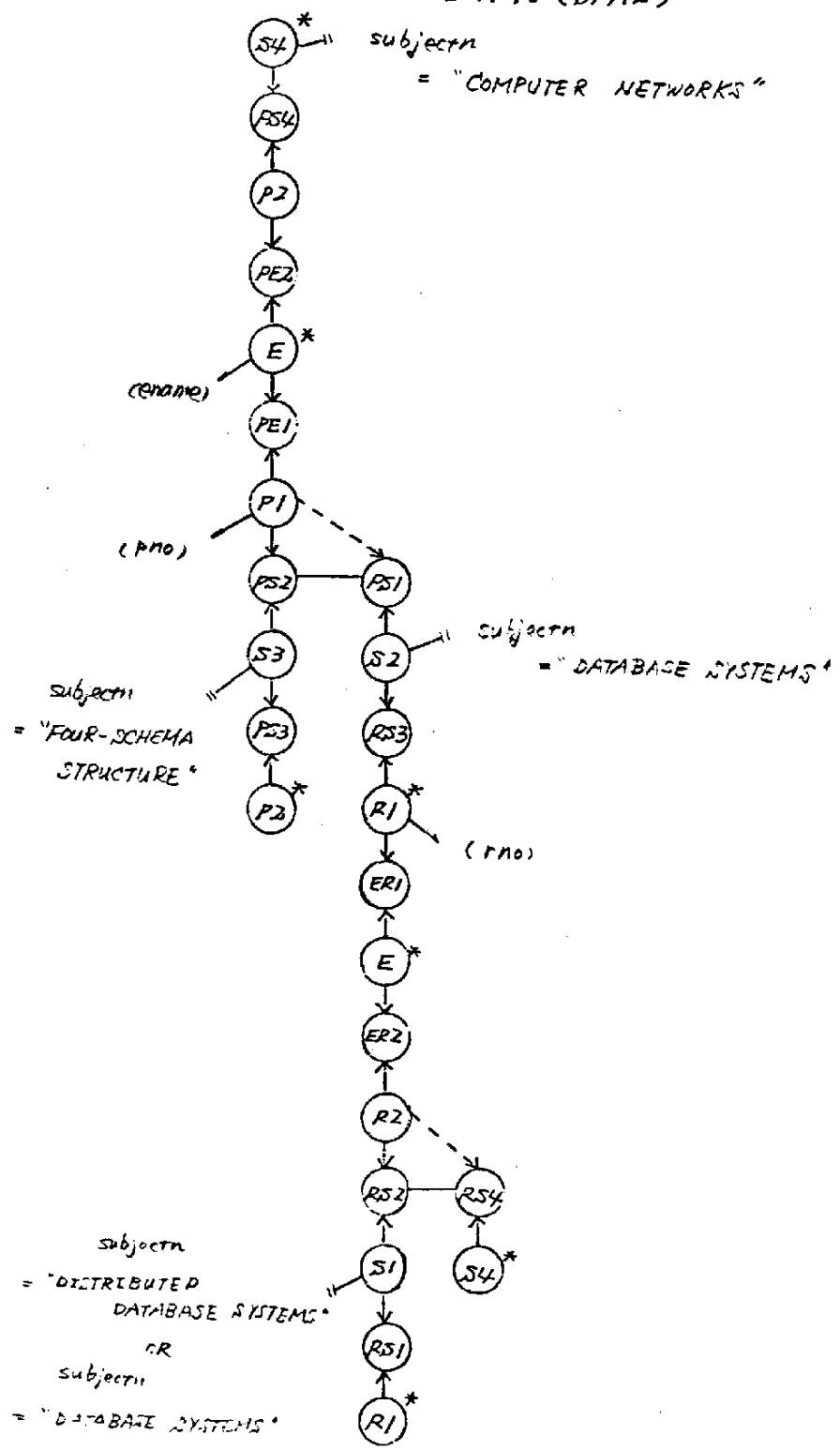


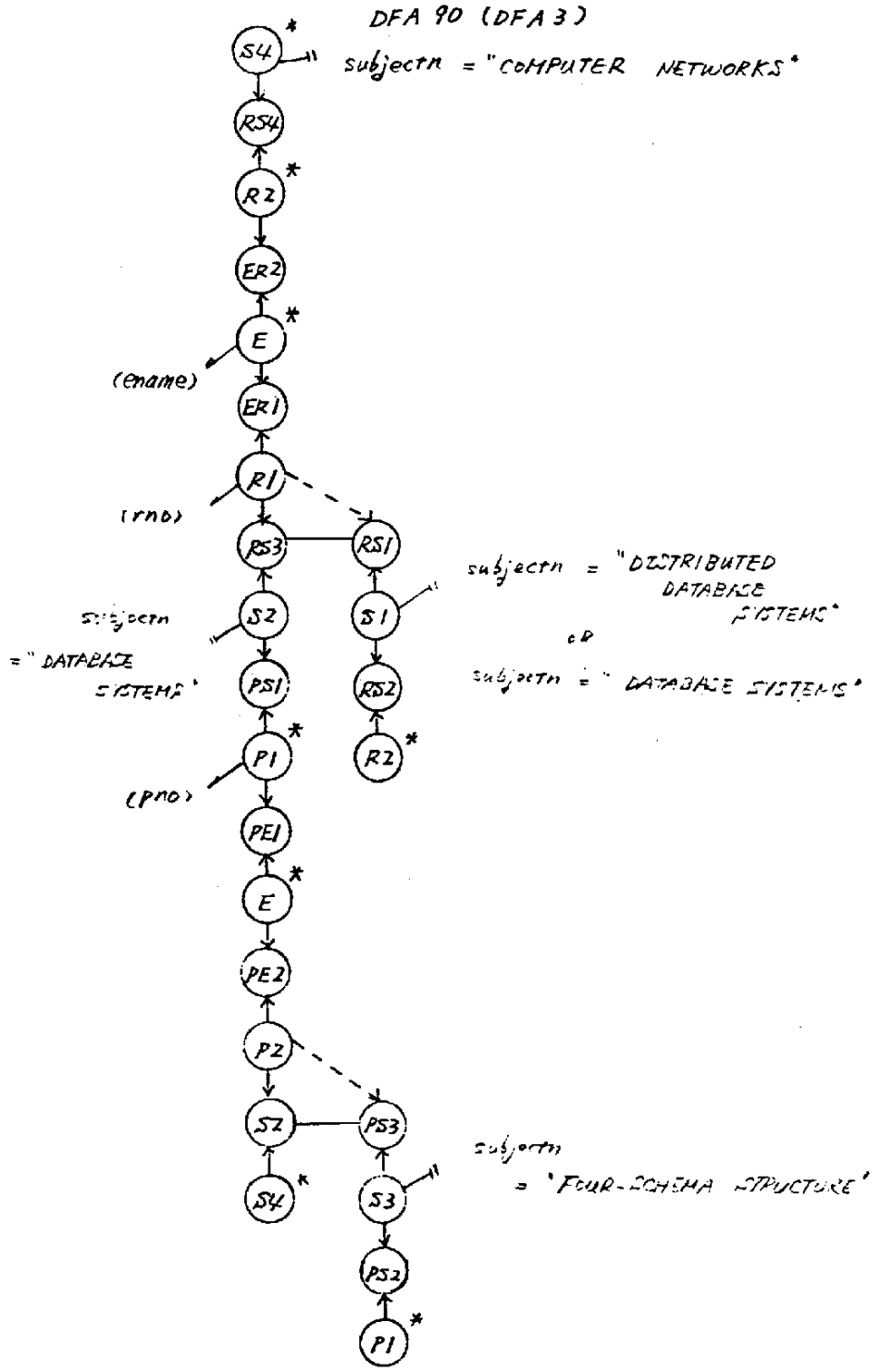

```
**DFA090
RANGE ( E, EMPLOYEE ) ( P1, PROJECT ) ( P2, PROJECT );
RANGE ( R1, REPORTR ) ( R2, REPORTR ) ( S1, SUBJECT );
RANGE ( S2, SUBJECT ) ( S3, SUBJECT ) ( S4, SUBJECT );

RANGE ( ER1, EMP-REP-LNK ) ( ER2, EMP-REP-LNK );
RANGE ( PE1, PROJ-EMP-LNK ) ( PE2, PROJ-EMP-LNK );
RANGE ( PS1, PROJ-SUBJ ) ( PS2, PROJ-SUBJ ) ( PS3, PROJ-SUBJ );
RANGE ( PS4, PROJ-SUBJ ) ( RS1, REP-SUBJ ) ( RS2, REP-SUBJ );
RANGE ( RS3, REP-SUBJ ) ( RS4, REP-SUBJ );
RETRIEVE  INTO   DFA090  ( E.ENAME, P1.PNO, R1.RNO )
WHERE
     (S1.SUBJECTN = "DISTRIBUTED DATABASE SYSTEMS"      OR
      S1.SUBJECTN = "DATABASE SYSTEMS"  )               AND
      S1.@J =  RS1.@J                                   AND
      RS1.@R  =  R1.@R       AND   R1.@R  =  ER1.@R     AND
      ER1.@E  =  E.@E        AND   E.@E   =  ER2.@E     AND
      ER2.@R  =  R2.@R       AND   R2.@R  =  RS2.@R     AND
      RS2.@J  =  S1.@J                                  AND
      R1.@R   =  RS3.@R      AND   RS3.@J =  S2.@J      AND
      S2.@J   =  PS1.@J                                 AND
      PS1.@P  =  P1.@P       AND   P1.@P  =  PE1.@P     AND
      PE1.@E  =  E.@E        AND
      S2.SUBJECTN   =  "DATABASE SYSTEMS"               AND
      P1.@P   =  PS2.@P      AND   PS2.@J =  S3.@J      AND
      S3.@J   =  PS3.@J                                 AND
      PS3.@P  =  P2.@P       AND   P2.@P  =  PE2.@P     AND
      PE2.@E  =  E.@E        AND   R2.@R  =  RS4.@R     AND
      P2.@P   =  PS4.@P      AND   PS4.@J =  S4.@J      AND
      RS4.@J  =  S4.@J                                  AND
      S4.SUBJECTN =  "COMPUTER NETWORKS"                AND
      S3.SUBJECTN =  "FAQ-SCHEMA STRUCTURE";
```

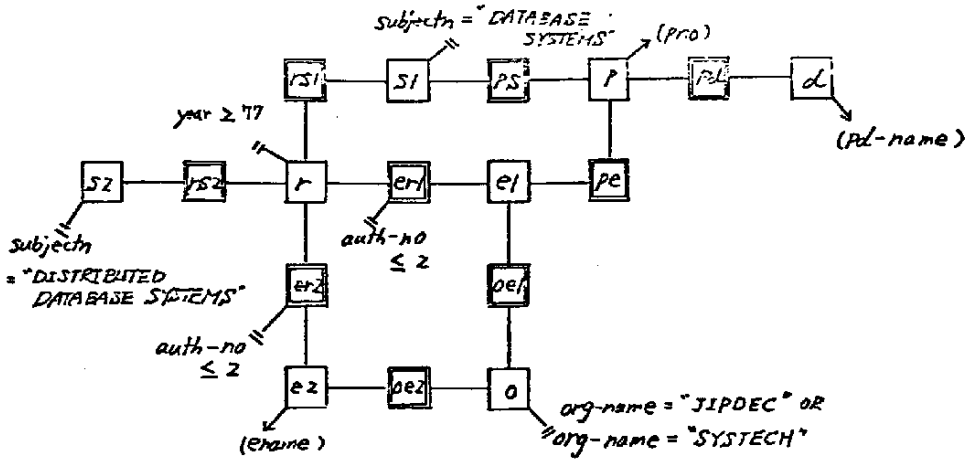

DFA 1 (DFA090)
S2
subjectn = "DATABASE SYSTEMS"
PS2
P1
(pno)
PE1
E
(ename)
PE2
P2
PS4
PS3
subjectn = "COMPUTER NETWORKS"
S4
S3
subjectn = "FOUR-SCHEMA STRUCTURE"
RS4
PS2
R2
P1
ER2
E
ERI
R1
(rno)
RS3
RS1
S2
S1
subjectn = "DISTRIBUTED DATABASE SYSTEMS"
OR
subjectn = "DATABASE SYSTEMS"
RS2
R2

DFA 90 (DFA2)


S4
subjectn
= "COMPUTER NETWORKS"
PS4
P2
PE2
E
(ename)
PE1
P1
(pno)
PS2
PS1
S3
S2
subjectn
= "DATABASE SYSTEMS"
subjectn
= "FOUR-SCHEMA
STRUCTURE"
PS3
PS3
P2
R1
(rno)
ER1
E
ER2
R2
RS2
RS4
subjectn
= "DISTRIBUTED
DATABASE SYSTEMS"
OR
subjectn
= "DATABASE SYSTEMS"
S1
S4
RS1
R1

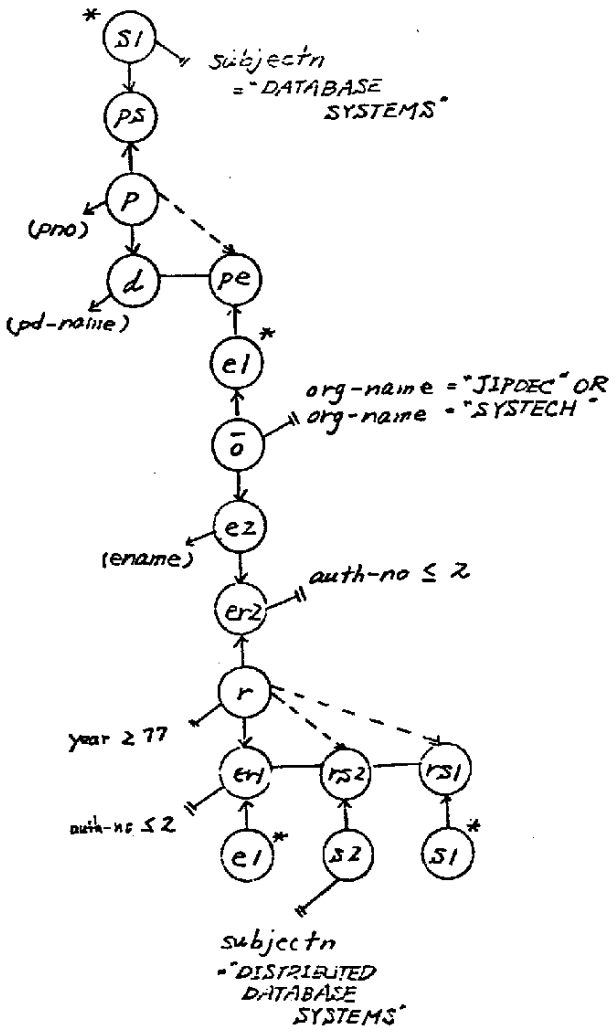

DFA 90 (DFA 3)
S4
subjectn = "COMPUTER NETWORKS"
RS4
R2
ER2
E
(ename)
ER1
R1
(rno)
RS3
RS1
subjectn = "DISTRIBUTED DATABASE SYSTEMS"
S2
S1
subjectn = "DATABASE SYSTEMS"
PS1
RS2
subjectn = "DATABASE SYSTEMS"
P1
R2
(pno)
PE1
E
PE2
P2
S2
PS3
S4
S3
subjectn = 'FOUR-SCHEMA STRUCTURE'
PS2
P1

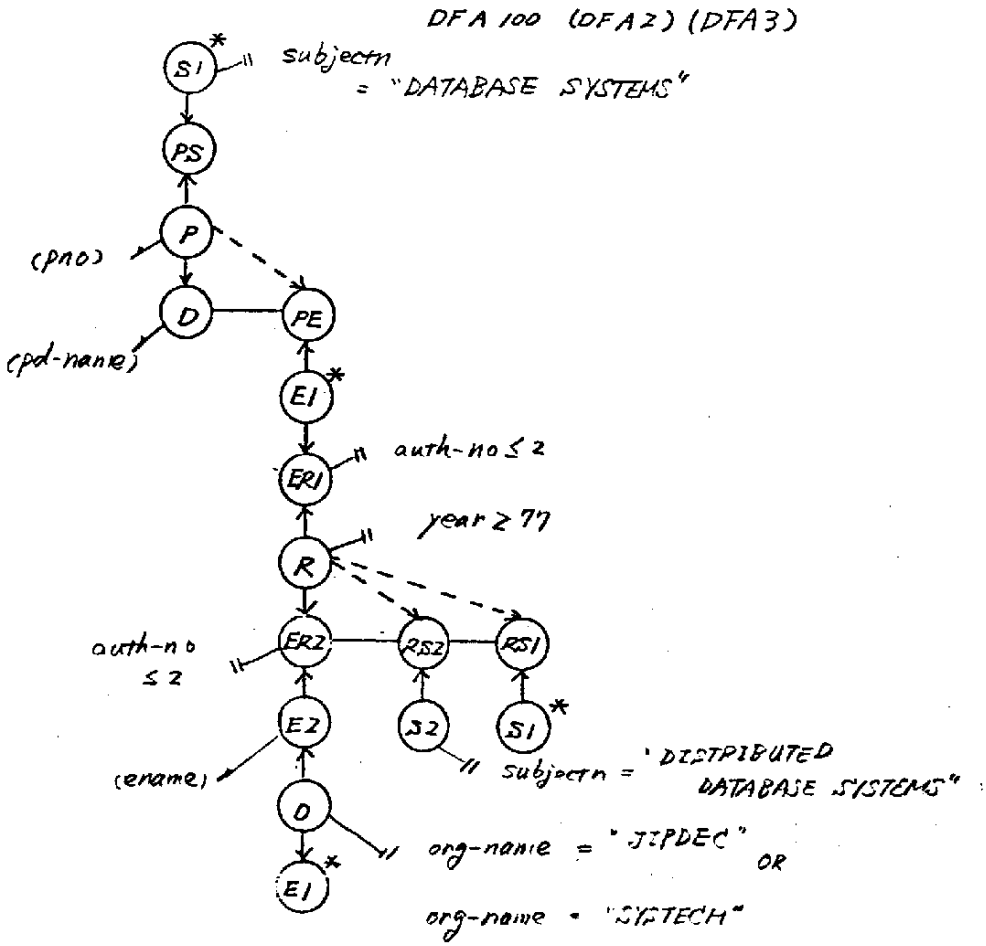
DFA 100



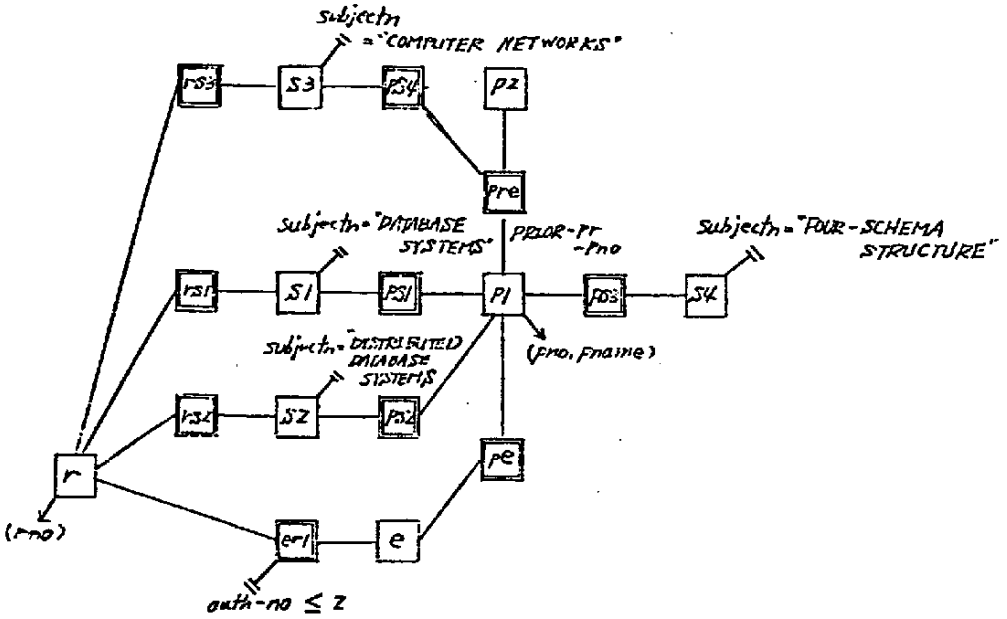

```
**DFA100
RANGE ( R, REPORTR ) ( S1, SUBJECT ) ( S2, SUBJECT );
RANGE ( E1, EMPLOYEE ) ( E2, EMPLOYEE ) ( O, ORGANIZATION );
RANGE ( P, PROJECT ) ( D, PRODUCT ) ( PD, PROJ-PROD );
RANGE ( RS1, REP-SUBJ ) ( RS2, REP-SUBJ ) ( PS, PROJ-SUBJ );
RANGE ( ER1, EMP-REP-LNK ) ( ER2, EMP-REP-LNK );
RANGE ( OE1, ORG-EMP ) ( OE2, ORG-EMP ) ( PE, PROJ-EMP-LNK );
RETRIEVE  INTO  DFA100    ( E2.ENAME, P.PNO, D.PD-NAME )
WHERE
    ( O.ORG-NAME = "JIPDEC"  OR  O.ORG-NAME = "SYSTECH" )          AND
      O.#ORGANIZATION = OE1.#ORGANIZATION                          AND
      O.#ORGANIZATION = OE2.#ORGANIZATION   AND   OE2.#E = E2.#E   AND
      E2.#E  =  ER2.#E    AND  ER2.AUTH-NO  LE  2                  AND
      ER2.#R =  R.#R      AND  R.YEAR       GE  77                 AND
      P.#R   =  ER1.#R    AND  ER1.AUTH-NO  LE  2                  AND
      ER1.#E =  E1.#E     AND  E1.#E        =  OE1.#E              AND
      E1.#E  =  PE.#E     AND  PE.#P        =  P.#P                AND
      P.#P   =  PD.#P     AND  PD.#PRODUCT  = D.#PRODUCT           AND
      P.#P   =  PS.#P     AND  PS.#J        = S1.#J                AND
      S1.SUBJECTN = "DATABASE SYSTEMS"                             AND
      S1.#J  =  RS1.#J    AND  RS1.#R       = R.#R                 AND
      R.#R   =  RS2.#R    AND  RS2.#J       = S2.#J                AND
      S2.SUBJECTN  = "DISTRIBUTED DATABASE SYSTEMS";
```

DFA 100 (DFA1)

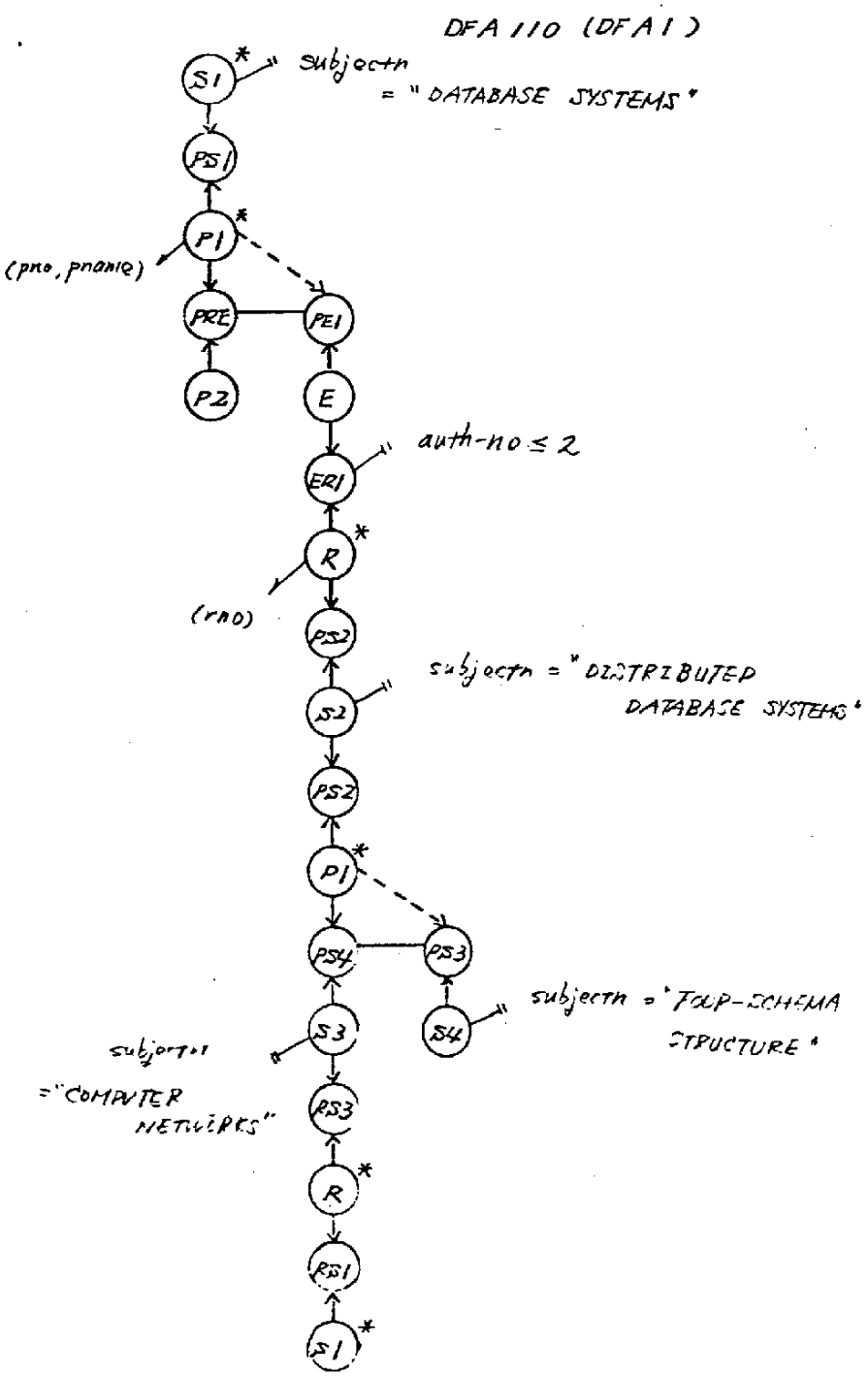

S1
subjectn
="DATABASE SYSTEMS"
PS
P
(pno)
d
pe
(pd-name)
e1
org-name = "JIPDEC" OR
org-name = "SYSTECH"
o
e2
(ename)
auth-no ≤ 2
er2
r
year ≥ 77
er1
rs2
rs1
auth-no ≤ 2
e1
s2
s1
subjectn
="DISTRIBUTED DATABASE SYSTEMS"


DFA 100 (DFA2) (DFA3)
S1
subjectn = "DATABASE SYSTEMS"
PS
P
(pno)
D
PE
(pd-name)
E1
ER1
auth-no ≤ 2
year ≥ 77
R
auth-no ≤ 2
ER2
RS2
RS1
E2
S2
S1
(ename)
subjectn = "DISTRIBUTED DATABASE SYSTEMS"
D
org-name = "JIPDEC" OR
E1
org-name = "SYSTECH"

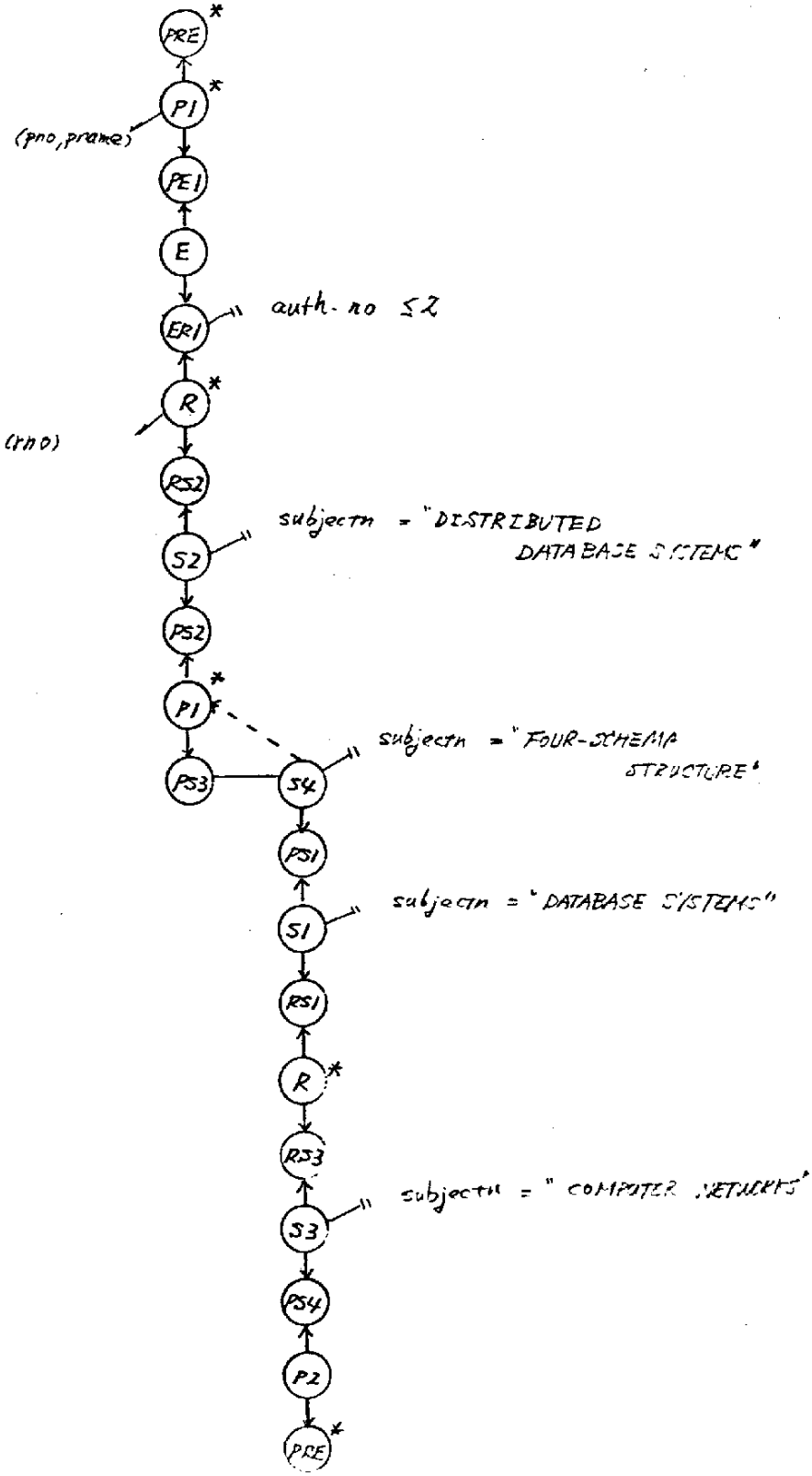
DFA 0110



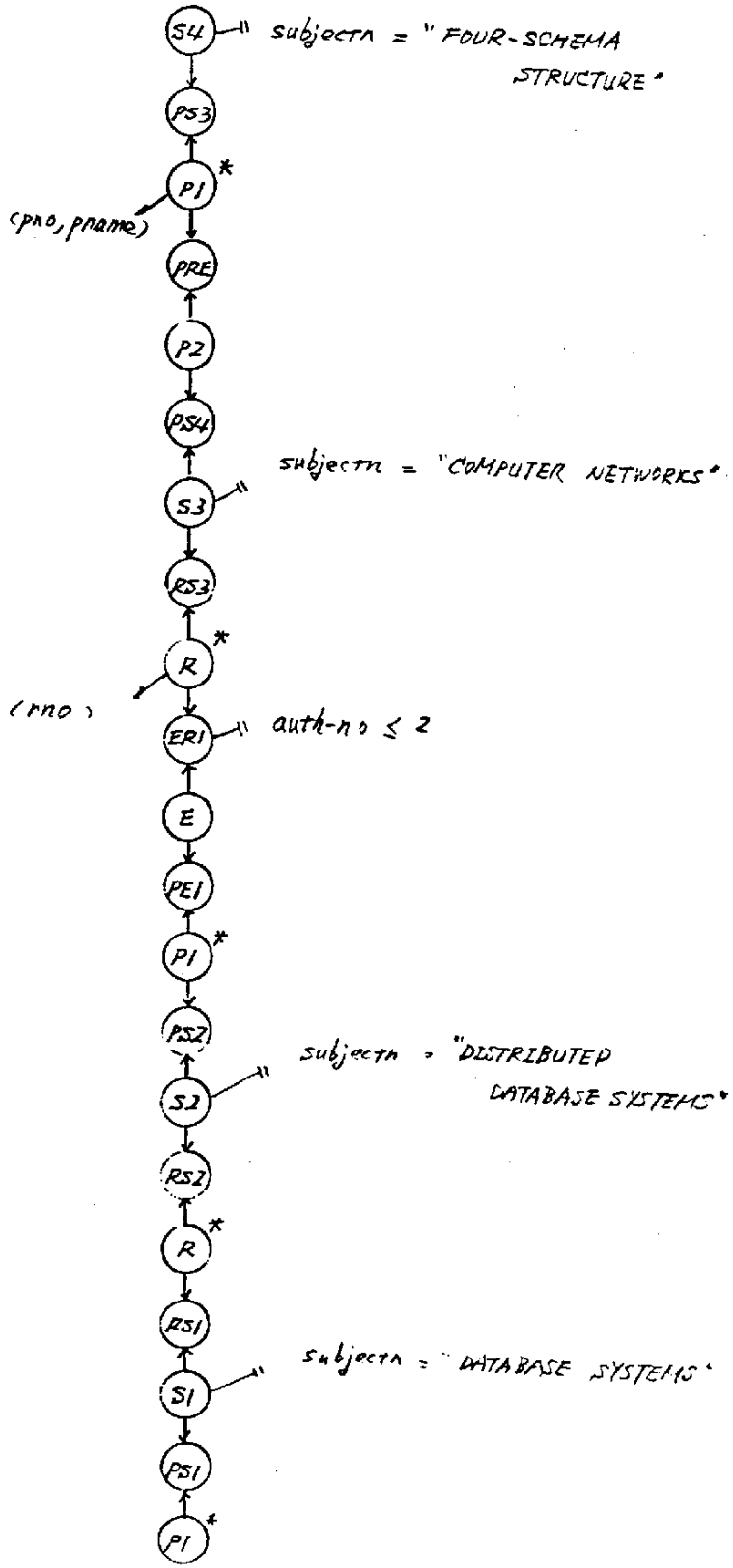

```
**DFA110
RANGE ( E, EMPLOYEE ) ( R, REPORTR ) ( P1, PROJECT ) ( P2, PROJECT );
RANGE ( S1, SUBJECT ) ( S2, SUBJECT ) ( S3, SUBJECT ) ( S4, SUBJECT );
RANGE ( ER1, EMP-REP-LNK ) ( PE1, PROJ-EMP-LNK ) ( PRE, PROJ-RE );
RANGE ( RS1, REP-SUBJ ) ( RS2, REP-SUBJ ) ( RS3, REP-SUBJ );
RANGE ( PS1, PROJ-SUBJ ) ( PS2, PROJ-SUBJ );
RANGE ( PS3, PROJ-SUBJ ) ( PS4, PROJ-SUBJ );
RETRIEVE INTO   DFA110    ( P1.PNO, P1.PNAME, R.RNO )
WHERE
      S4.SUBJECTN = "FOUR-SCHEMA STRUCTURE"                          AND
      S4.#J  =  PS3.#J            AND   PS3.#P  =  P1.#P    AND
      P1.#P  =  PRE.PRIOR-PR-#P   AND   PRE.#P  =  P2.#P    AND
      P2.#P  =  PS4.#P            AND   PS4.#J  =  S3.#J    AND
      S3.SUBJECTN = "COMPUTER NETWORKS"                              AND
      S3.#J  =  RS3.#J            AND   RS3.#R  =  R.#R     AND
      R.#R   =  ER1.#R            AND   ER1.#E  =  E.#E     AND
      E.#E   =  PE1.#E            AND   PE1.#P  =  P1.#P    AND
      P1.#P  =  PS1.#P            AND   PS1.#J  =  S1.#J    AND
      S1.#J  =  RS1.#J            AND   RS1.#R  =  R.#R     AND
      S1.SUBJECTN = "DATABASE SYSTEMS" AND R.#F =  RS2.#R   AND
      RS2.#J =  S2.#J             AND   S2.#J   =  PS2.#J   AND
      P1.#P  =  PS2.#P            AND
      S2.SUBJECTN = "DISTRIBUTED DATABASE SYSTEMS"                   AND
      ER1.#E =  P1.#E             AND   ER1.AUTH-NO  LE 2;
```


DFA 110 (DFA 1)
S1
*
subjectn
= "DATABASE SYSTEMS"
PS1
P1
*
(pno, pname)
PRE
PE1
P2
E
auth-no ≤ 2
ER1
R
*
(rno)
PS2
subjectn = "DISTRIBUTED
DATABASE SYSTEMS"
S2
PS2
P1
*
PS4
PS3
S3
S4
subjectn = "TOP-SCHEMA
STRUCTURE"
subjectn
= "COMPUTER
NETWORKS"
RS3
R
*
RS1
S1
*

DFA 110 (DFA 3)


PRE
P1
(pno, prname)
PE1
E
ER1
auth-no ≤ 2
R
(rno)
RS2
S2
subjectn = "DISTRIBUTED DATABASE SYSTEMS"
PS2
P1
PS3
S4
subjectn = "FOUR-SCHEMA STRUCTURE"
PS1
S1
subjectn = "DATABASE SYSTEMS"
RS1
R
RS3
S3
subjectn = "COMPUTER NETWORKS"
PS4
P2
PRE

DFA 110 (DFA2)


S4
subject = "FOUR-SCHEMA STRUCTURE"
PS3
P1
*
(pno, pname)
PRE
P2
PS4
S3
subject = "COMPUTER NETWORKS"
RS3
R
*
(rno)
ER1
auth-no ≤ 2
E
PE1
P1
*
PS2
S2
subject = "DISTRIBUTED DATABASE SYSTEMS"
RS2
R
*
RS1
S1
subject = "DATABASE SYSTEMS"
PS1
P1
*

— 禁 無 断 転 載 —

昭和５７年３月発行

発行所　財団法人　日本情報処理開発協会
東京都港区芝公園３－５－８
機械振興会館内
ＴＥＬ（434）8211（代表）

印刷所　株式会社　昌文社
東京都港区芝５－２６－３０
全専売ビル
ＴＥＬ（452）４９３１

56-S002

原本（持出厳禁）

受　付 No.	D-15
受付年月日	57. 3. 31
作　成　課	開発第2課