

57—S 001

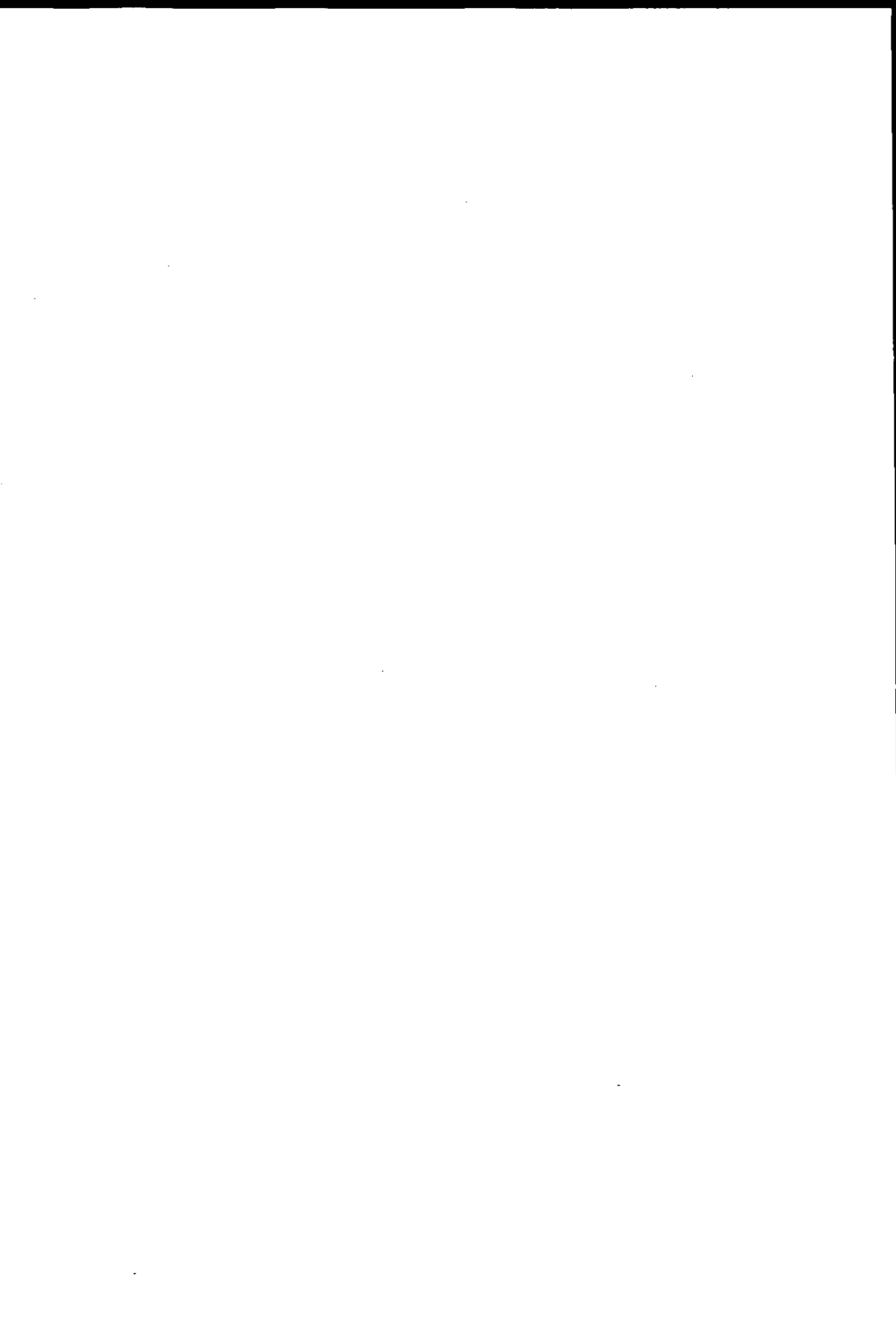
マン・マシン・ユーザ・インタフェース
に関する調査研究報告書

昭和 58 年 3 月



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57年度に実施した「マ
ン・マシン・ユーザ・インタフェースの調査研究」の成果をとりまと
めたものであります。



序 文

今後の情報処理に要求される基本的条件の1つは、より広汎な分野及び、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・システムの利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。

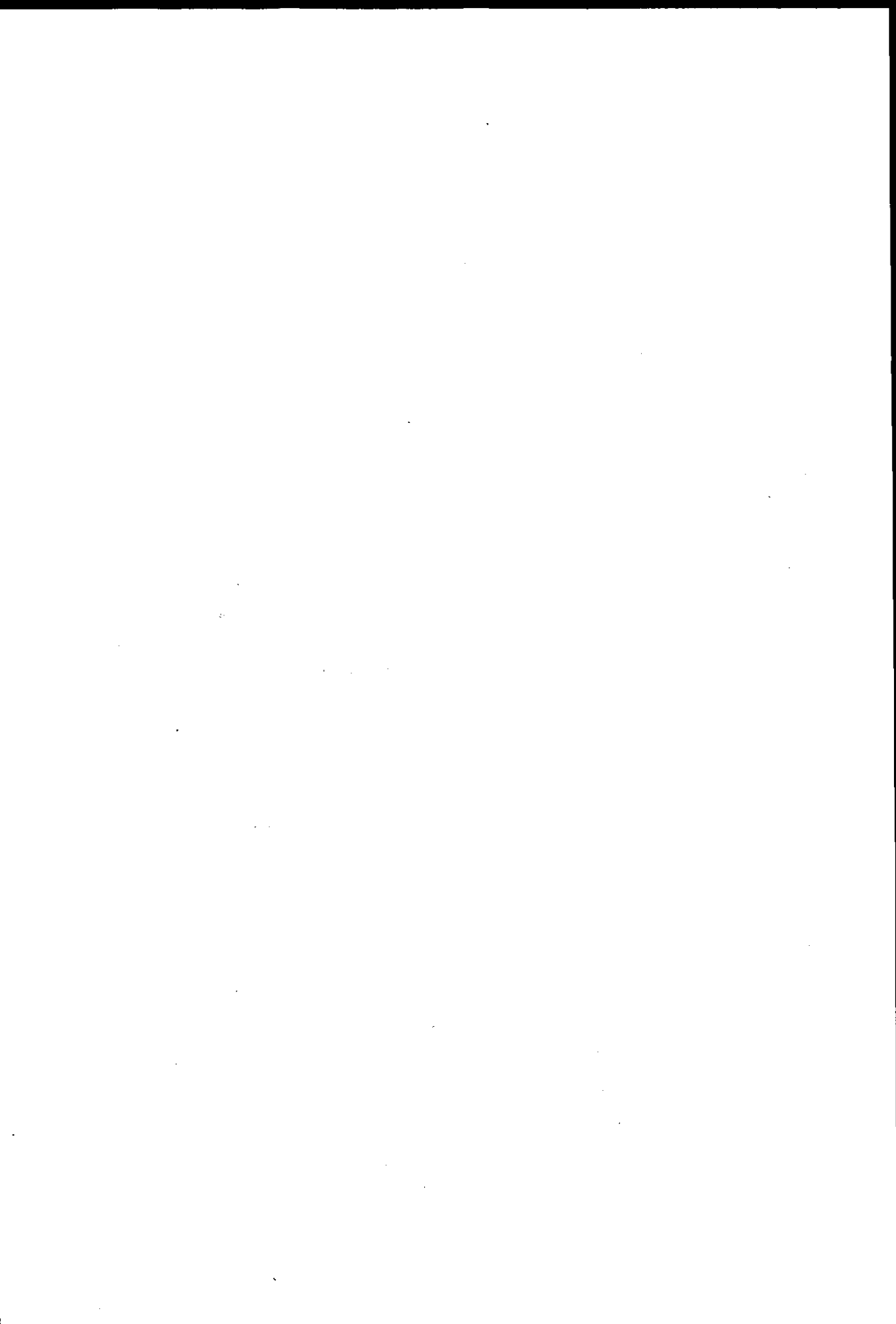
情報処理システムが多くの非専門家にとって使い易いツールとなるには、コンピュータ側のインタジェンスを向上させ、人間の五感による情報伝達機能により近い形態で各種の情報の処理が容易に行なえることが必要不可欠となる。

当協会では、昭和55年度より3ヶ年の計画で「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、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、実現を期待されるマン・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、特にその中で最も重要と思われる分散型データベースを中心にして調査研究を行ってきた。

本報告書はその最終年度の成果、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実現結果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58年 3 月



目 次

序 文	
概 要	1
第 I 部 分散型データベースシステム J D D B S の全体アーキテクチャ	3
1. 概 説	3
2. 研究開発動向	4
3. J D D B S の全体アーキテクチャ	6
3.1 四層構造	6
3.1.1 四 階 層	6
3.1.2 D D B S 設計	6
3.1.3 D D B S の運用	8
3.1.4 共通モデル	9
3.2 D D B S のシステム構成	9
第 II 部 異種性問題	13
1. データベースシステム (D B S) の異種性	13
2. 概念モデル	14
2.1 データモデルの概念化	14
2.2 概念データ構造	14
2.3 概念操作言語	15
3. CODASYL データベースシステム	17
3.1 CODASYL D B S の階層構造	17
3.2 論理 CODASYL モデル	18
3.2.1 論理データ構造	18
3.2.2 論理操作演算 — 検索	20
3.2.3 論理操作演算 — 更新演算	26
3.3 仮想 CODASYL D B S	32
3.3.1 仮想データ構造	32
3.3.2 仮想 CODASYL マシン (V C M)	34
3.4 概念 CODASYL モデル	43
3.4.1 概念データ構造	43
3.4.2 基本操作演算	44
3.5 物理 CODASYL モデル	46

3.5.1	物理データ構造	46
3.5.2	物理操作演算	48
4.	概念モデルによるCODASYLモデル表現	49
4.1	LCモデルのデータ構造	49
4.2	LCモデルの操作演算	51
4.2.1	LCモデルの基本演算	51
4.2.2	LC操作言語(LCML) - RQL検索	52
4.2.3	LC操作言語(LCML) - RQL更新	56
5.	LCMLのCQL表現	61
6.	CODASYL問合せのDML表現	64
6.1	CQGのDMLによる表現	64
6.2	出力テープの最適化	67
6.3	DML生成アルゴリズム	78
7.	CQGの巡航木表現方法	86
7.1	目標と仮定	86
7.2	アクセスコストと出力コスト	88
7.3	コストの見積り	89
7.4	巡航木の期待アクセスコスト	98
7.5	巡航木の期待出力コスト	100
7.6	巡航木生成アルゴリズム	101
8.	ローカルデータベースプロセッサ(LDP)	107
9.	LDPの性能と利用性の評価	109
9.1	性能評価	109
9.1.1	評価環境	109
9.1.2	各構成モジュールの負荷	109
9.1.3	アクセスパス生成の負荷	111
9.1.4	アクセスパス生成	111
9.1.5	COBOL DMLプログラム	115
9.2	利用性評価	116
9.2.1	評価方法	116
9.2.2	問題	117
9.2.3	評価結果	117
9.3	まとめ	118
10.	まとめと今後の課題	119
第Ⅲ部	分散性問題	121

1. 統合化	122
1.1 GCデータ構造	122
1.2 GC演算	123
1.3 GCデータ構造定義	124
1.4 分散情報 (DI)	126
2. 分散問合せ処理 (DQP)	127
2.1 表現変換 (ETP)	127
2.2 ILQP	129
2.3 DM間問合せ処理 (DMQP)	130
2.3.1 DM間問合せ (DMQ)	130
2.3.2 単純問合せ	131
2.3.3 ビットマップ応答の利用ーアルゴリズムBBS	134
2.3.4 一般問合せ	137
2.3.5 制限方法による一般問合せ処理	140
2.3.6 同定方法による一般問合せ処理	141
2.4 集積関数 (aggregate function) の処理	154
3. 分散更新処理 (DUP)	160
3.1 コミットメント制御	160
3.1.1 2相コミットメント制御	161
3.1.2 放送2相コミットメント制御	161
4. 全体データベースプロセッサ (GDP)	164
5. まとめと今後の課題	166
第IV部 JDDBSの実現	167
1. JDDBSの全体構成	167
1.1 データベースシステム (DBS)	169
1.1.1 AIM DBSの特徴	169
1.1.2 DCDBS (ドキュメントDBS)	171
1.1.3 PRDBS (プロジェクトDBS)	179
1.2 システム格納領域 (SS)	187
2. ローカルデータベースプロセッサ (LDP-V3)	204
2.1 LDP-V3のシステム構成	204
2.2 HIPの実現	205
2.3 QTPの実現	235
2.3.1 単純集積関数	237
2.3.2 group-by集積関数	238

2.3.3	G-tree	244
2.3.4	QTPの実現	247
2.3.5	結果関係の管理	277
2.3.6	COBOL DMLの起動方法	281
2.3.7	LDPの同時実行制御	281
3.	全体データベースプロセッサ (GDP)	285
3.1	全体データベースプロセッサ (GDP)	285
3.1.1	機能概要	285
3.1.2	GDPの構成	286
3.1.3	GDPコマンドクラス	288
3.2	DIPの実現	288
3.2.1	DIPの概要	288
3.2.2	DIの定義と実現	289
3.2.3	GCS定義式の内部表現変換アルゴリズム	291
3.2.4	GCS定義例	303
3.3	DQPの実現	305
3.3.1	DQPの概要	305
3.3.2	DQPの実現	306
3.3.3	メッセージフォーマット	309
3.3.4	UST/DMQPの状態遷移	319
3.3.5	モジュール間通信の実現	338
3.3.6	DMQP間結合処理の実現	348
3.3.7	GCS問合せの実行	355
4.	JDDBSの実働例	358
5.	まとめと今後の課題	371
	参考文献	372
付記 1	LDP コマンド	377
付記 2	GDP コマンド	422
付記 3	LDP 使用手引	442
付記 4	DML I	459
付記 5	集積関数処理の評価	519
付記 6	発表論文リスト	535
付記 7	英文資料	539

概 要

マン・マシン・ユーザ・インタフェースは多様な側面からとらえることができる。例えば、従来はTSSにおける会話型言語の機能や、コマンドの操作性、あるいは端末のキーボード配列やコントロール・キーの位置等、利用者が直接マシンあるいはシステムに触れる部分のみをクローズアップしたマン・マシン・ユーザ・インタフェース論が中心であった。

本プロジェクトでは、コンピュータ・システムが今後更にすぐれた人間のアシスタントとなるべきであるという観点から1985年以降を目途とした、より将来的、且つ理想的なマン・マシン・ユーザ・インタフェースに改革的な変化を与え得るために実現されねばならぬ基本的な情報処理技術要素自体に体する検討を行うことを主目標とした。本調査研究は昭和55年度より3ヶ年計画で実施され、本年度はその最終年度にあたる。

第1年度は、マン・マシン・ユーザ・インタフェースを支える今後の新しい情報技術として、音声、画像、図形、文字、記号等各種情報メディアの統合処理を可能とするファイル、入力装置等の技術動向を調査するとともに、オフィス情報システム(OIS)等の今後の新情報システムの中核技術としての分散型データベースシステム(DDBS)の技術的検討を行った。

第2年度は、第1年度の検討をもとにして、今後のマン・マシン・ユーザ・インタフェース・システムの中核技術としての分散型データベースシステム(DDBS)の1部のモデル化を行った。種々のデータベースシステム(DBS)を共通的にアクセスする為の高水準共通インタフェースシステムとしてのローカルデータベースプロセッサ(LDP)を実現した。また性能面、利用面からの評価を行い、LDPの有用性を明らかにした。LDPは、CODASYL DBSに対して、利用者は、ほぼ完全な関係操作演算(検索更新)を述語論理形式の非手続的言語RQLで記述することによってDBSを操作できる。

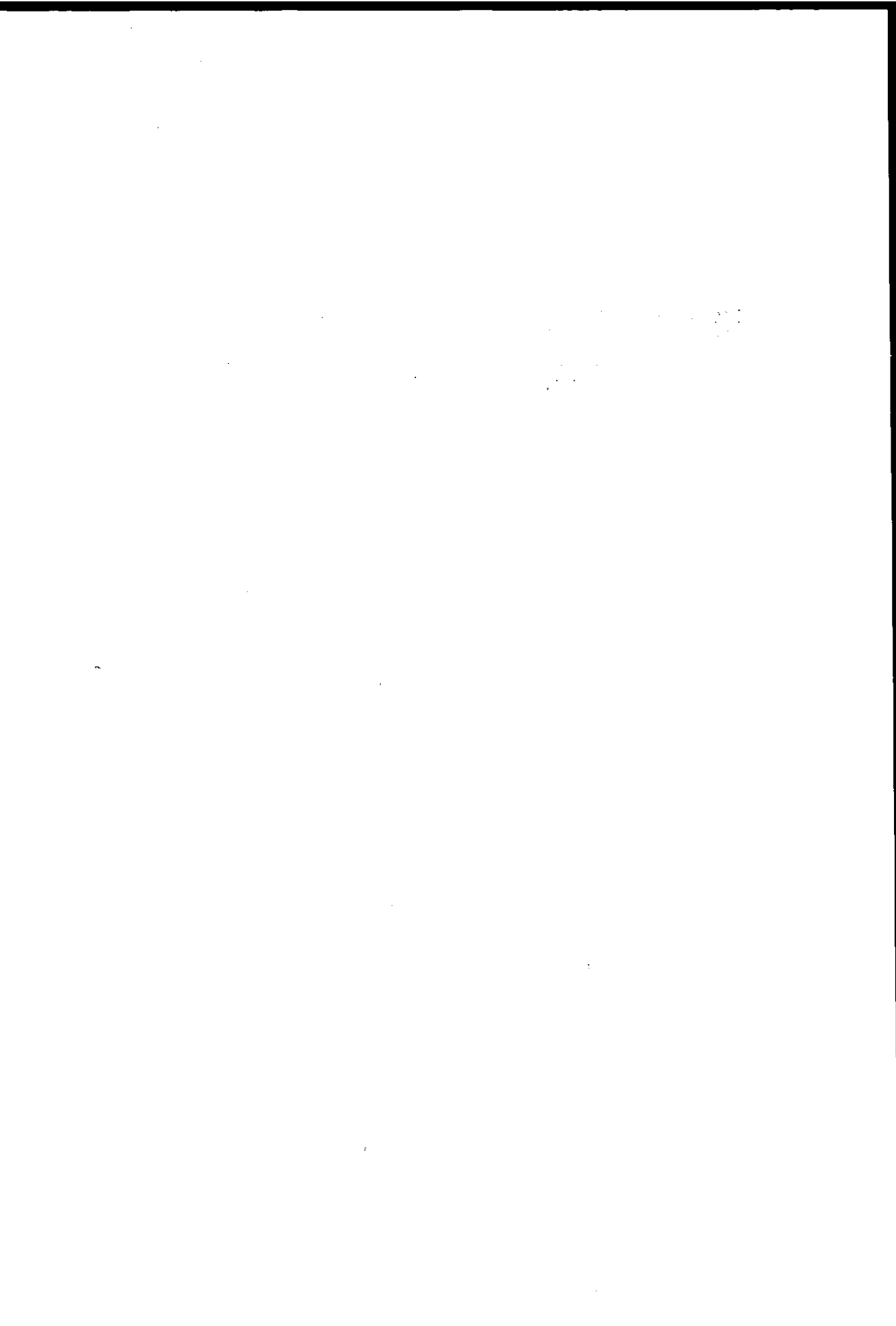
最終年度である今年度は、LDP上に、通信処理用の全体データベースプロセッサ(GDP)を実現し、異種分散型データベースシステムJDDBSの実働と評価を行った。利用者は、どこにどのデータが、どの様な形式で存在するかを意識することなく、あたかも1つの関係データベースシステムかの様に関係RQL問合せを記述できる。JDDBSは、この問合せを、各DBSに分割し、LDPでCODASYL DBSから解を出し、GDPの通信処理によって、DBS間の問合せを処理出来る。現在JDDBSは、同一ホスト内の複数CODASYL DBSから構成されているが、異種DBSを、DBS側から必要となる基本機能を実現できたと考えている。本報告書は、3年間の調査研究の最終報告書として、JDDBSの基本概念、実現方式及び実現結果とその評価についてまとめている。

第I部では、DDBSの定義とJDDBSの全体アーキテクチャを述べている。第II部では、DDBSを構成するDBSの異種性をデータモデルの相違としたときの解決方法として、特にCODASYL DBSに着目してその解法及びその正しさを示している。第III部では、DDBSが複数のDB

Sから成ることから生じる分散性問題の解法を示している。第Ⅳ部では、主にJ D D B Sの実現面で用いた方式について述べている。

なお、本文中で、同一部内の章、図表を参照するときは、章、図表番号のみで表し、他の部の章、図表を参照する場合は、部番号を付加して表している。

第I部 分散型データベースシステム JDDBSの全体アーキテクチャ



第 I 部 分散型データベースシステム JDDBS の全体アーキテクチャ

1. 概 説

種々の情報システムにおいて、データベースシステム (DBS) が中核的な存在を占めてきている。この背景には、データモデル、ANSI / X 3 / SPARC [TS 1 C D 7 8] をはじめとする DBS 技術の進歩がある。一方、ARPANET、CYCLADES、Ethernet 等の通信網の進歩は、計算機システム間での相互通信を可能としてきている。VLSI によるハードウェアの進歩によって、数 MIPS もの処理能力を有したパーソナル計算機が現われ出し、各利用者の必要とする処理能力を各人が専有出来る様になってきている。この様な時、複数の利用者によって共有されるべき計算機資源は、データベースであり、種々の通信網を通して複数の種々のデータベースを共有する問題が重要となってくると思われる。オフィス情報システム (OIS)、テレビ会議、意思決定支援システム、CAD において、複数データベースの共有技術、即ち分散型データベースシステム (DDBS) 技術が中核技術となると考えられる。

DDBS とは、意味的に関連した複数のデータベースシステム (DBS) を、通信網を用いて結合し、利用者に対して仮想的な 1 つの DBS サービスを提供するシステムと定義出来る。ここで DBS とは、あるデータモデル (例えば、関係モデル [CODDE 7 0]、CODASYL モデル [CODAS 7 3]) を利用者に提供するオートマトンとする。データモデルとは、データ構造と、この上の操作演算とを意味するとする。よって、DBS の異種性 (相違) とは、各データモデルの相違と考えられる。他に、DBS の基づく OS、DBMS の機能 (例えば、インテグリティとセキュリティ機能)、DBS の存在する計算機ハードウェア等の相違が実際には存在するが、ここでは、最も本質的なデータモデルの相違について考えることにする。異種の DBS から成る DDBS を、異種 DDBS と呼ぶことにする。異種 DDBS を実現する為には、各 DBS の異種性 (即ち、各データモデルの相違) と、各 DBS が通信網上に分散していることによって生じる分散性との 2 つの問題解決が必要となる。本章では、異種性と分散性との 2 つの問題の解決技法について述べる。

2. 研究開発動向

第1章で定義した様な分散型データベースシステム (DDBS) の研究開発は、主に2つの時期に分けて考えられる。第I期研究開発は1976年頃から、1980年頃までに行われ、代表例としては、米国のCCA (Computer Corporation of America) 社のSDD-1 [ROTHJ80] 仏国のグルノーブル大学のPOLYPHEME [ADIBM80] の2つをあげることが出来る。これ等の特徴は、次の様である。

- (1) 同種の関係DBSから構成されている。

SDD-1はData computerであり、POLYPHEMEはURANVSである。

- (2) 通信網としては、ARPANET (SDD-1)、CYCLADES (POLYPHEME) といった広域パケット交換網を用いている。

- (3) 利用者は、データの所在を意識することなく (分散の不可視性) 検索問合せを行うことができる。利用者による問合せをDDBSが分散処理をして、利用者に解をわたす。

他に、冗長及び非冗長なデータを有した複数DBSへの更新演算の同期手法についての理論的研究がなされた [BERNP80]。この時期のDDBSは、複数の関係DBSをあたかも1つのDBSの様に利用者に見させ、データの所在と独立に関係問合せを記述させ、その分散処理機能を実現し、DDBSの可能性を示したと言える。これらの第I期DDBSの主要な問題点は、使用した広域通信網の低速性 (≤ 50 Kbps) によって、網がパフォーマンスのボトルネックとなったことである。

第II期DDBSの研究開発は、1980年頃から1985年頃の完成を目指して行われている。この例として、仏のINRIAのSIRIUS-DELTA [LEBIJ80]、グルノーブル大学のMICROBE-SCOT [TOANN81]、CCA社のMULTIBASE [SMITJ81]、IBMのSYSTEM D [ANDLS82]、及び国内では沖電気のBANET [YAMAH82]、我々のJDDBS [TAKIM78-93] がある。この第II期DDBSの特徴を次に示す。

- (1) 異種のDBSを構成要素とする。JDDBS、MULTIBASEでは、関係DBSに加えて、CODASYL DBSを構成要素としている。

- (2) 従来の広域網に加えて、Ethernetの様なローカルエリア網 (LAN) を用いる。

SIRIUS-DELTA、MICROBE-SCOT、SYSTEM D、沖電気BANET、JDDBS がこの例である。

- (3) 従来のDBMSによって管理された大型DBSに加えて、パーソナル計算機を主要な構成要素とする。

- (4) オフィス情報システム (OIS) への適用を試みている。

- (5) 検索の分散処理に加えて、更新の分散処理 (同時実行制御、コミットメント制御、障害回復、セキュリティ制御) の実現を行う。

第II期では、第I期DDBSに対して、異種DBSの組み込みと共に、高速、高信頼なLANを用いて通信のボトルネックの解消を目指している。Ethernetの様な通信媒体を共有した形態のLANの

他の大きな特徴は、従来の広域網に対して、1対多（放送）通信機能を物理的にサポートしている点である。この機能は、第Ⅲ部で見る様に、DDBSの通信処理を有効にかつ簡単化する。更に大きな特徴は、パーソナルDBSの様な小型DBSを主要な構成要素としてきている点である。個人ベースで生成してきた種々の形式のデータ（手紙、文書、図、音声等）を、どの様に共有していくかは、オフィス情報システム（OIS）を考えるうえでも重要な今後の課題である。

3. J D D B S の全体アーキテクチャ

本章では第1章で述べたDDBS実現上の異種性と分散性との2つの問題を解決する為にJDDBSで用いられている全体アーキテクチャについて論じる。

3.1 四層構造

3.1.1 四階層

第1章で定義した様にDDBSは、種々のDBSを通信網によって結合し、利用者にあたかも1つのDBSかの様に見えるシステムである。この為には、各DBSの異種性（即ち、各データモデルの差）と各DBSの分散性を解決する必要がある。この2つの問題を独立して解決する為に、図3.1に示す様な4つの階層を考える。ローカル内部（LI）層は、既存DBSに対応し、DBSがDDBSを通して利用可能なデータ構造を表すローカル内部スキーマ（LIS）と操作言語（LIML）とを提供している。LISとLIMLとは、各DBS固有のデータモデル（e.g. 関係モデル、CODASYLモデル）に基づいている。LISは、既存DBSのスキーマ、又は副スキーマに対応している。

ローカル概念（LC）層は、DDBS全体で共通なデータモデル（共通モデルとする）によるデータ構造を表すLCスキーマ（LCS）とLC操作言語（LCML）とから成る。LCSは、LISを共通モデルによって表現したものである。この層では、利用者は各DBSの異種性（データモデルの差）を意識することなく、DDBS全体で共通なデータモデルのもとで、共通な言語LCMLを用いてLCデータ構造の操作を行える。即ち、異種問題が解決されている。

全体概念（GC）層は、DDBS全体の一意なデータ構造記述としての全体概念（GC）スキーマ（GCS）と、この上のGC操作言語（GCML）とから成る。GC層は、LC層と同一のモデルに基づいている。この層では、利用者は、各DBSの存在（即ち、分散性）を意識することなく、DDBSをあたかも1つのDBSかのように見ることが出来る。よって、このGC層において異種性に加えて分散性問題も解決されている。

外部（EX）層は、DDBSの各応用固有のデータ構造としてのEXスキーマ（EXS）とEX操作言語（EXML）とから成る。EX層は、DDBSが既にGC層で異種性と分散性が解決され、1つのDBSに仮想化されているので、DBSのEX層と同一と考えられる。以降、LIS、LCS、GCSについて主に論じることとする。

3.1.2 DDBS設計

DDBSの設計方法としては、まず既存DBSの存在から出発してDDBSを実現する統合型設計と、1つの仮想DBSから出発して、これを複数DBSで実現する分割型設計とがある。前者では、各DBSのLISがまず存在して、このようにLCSを定義し、更にLCSの集合上にGCSが定義され、ついでEXSがGCS上に定義されるという設計段階がとられる〔図3.2〕

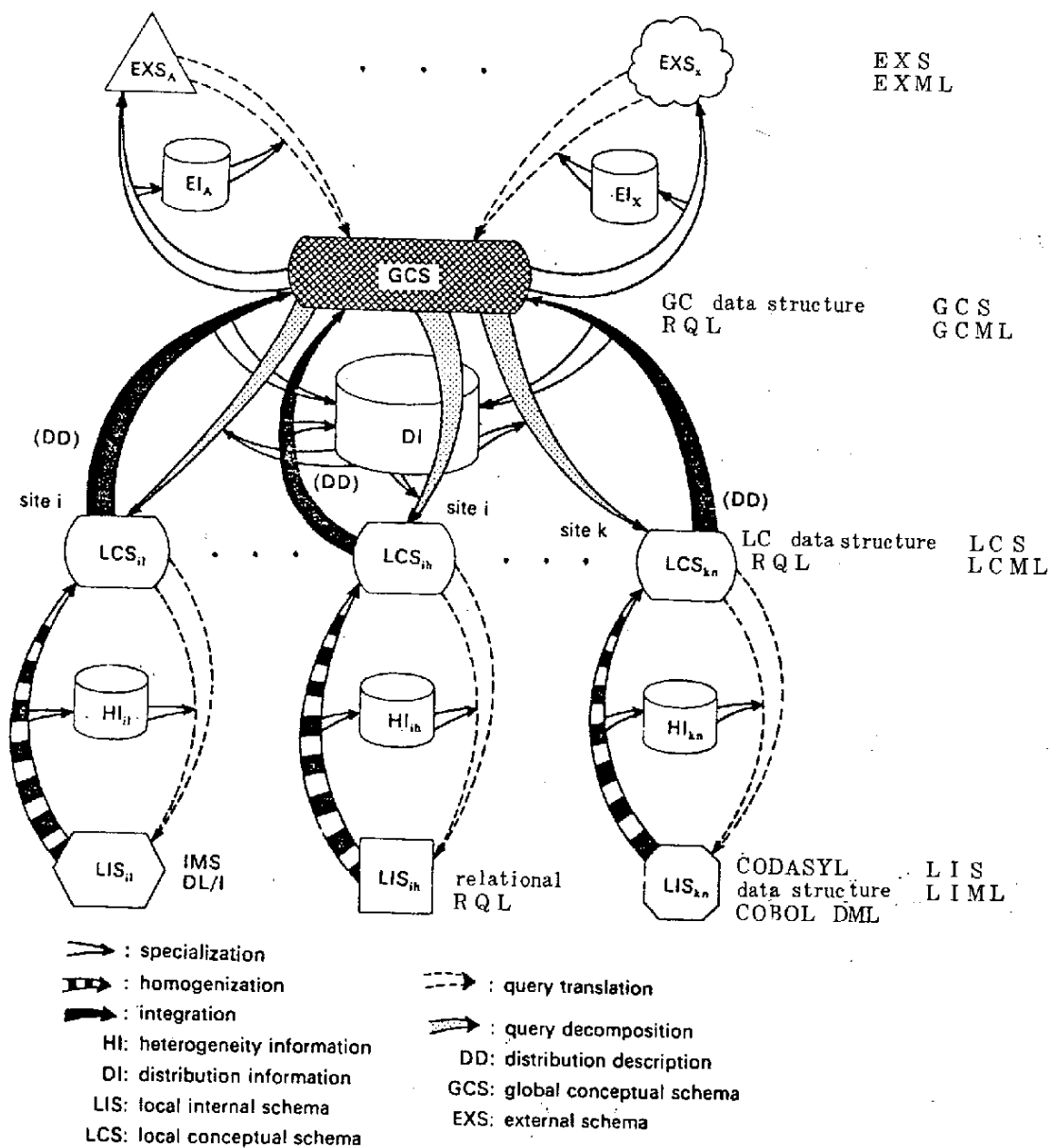


图 3.1 四 層 構 造

第一段階のLISからLCSを共通モデルによって定義することを同種化とする。次のLCSからGCSを定義することを統合化とする。最後のGCSからEXSの定義を特殊化とする。同種化では、LCSの記述と、LCSとLISの対応情報が、異種性情報(HI)として生成される。統合化では、GCSの記述と、LCSとGCSの対応情報が、分散情報(DI)として生成される。特殊化では、EXSと、GCSとEXSの対応情報が外部情報(EI)として生成される。これらのHI、DI、EIをまとめてDDBSのDD/D(データ辞書/ディレクトリ)とする。HIは、各DBS毎に保持され、DIは各DBSが完全コピーしたものを保持し、EIは応用の存在するDBSで保持される。我々のJDDBSは、統合型設計に基づいている〔図3.1〕。

一方、分割型設計の場合には、今述べたとは逆に、EXS、GCS、LISの順に定義される。

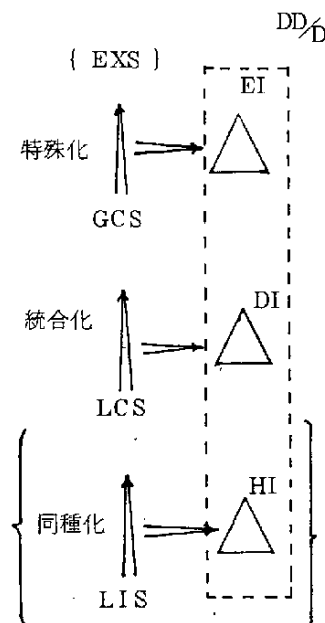


図 3.2 統合型設計

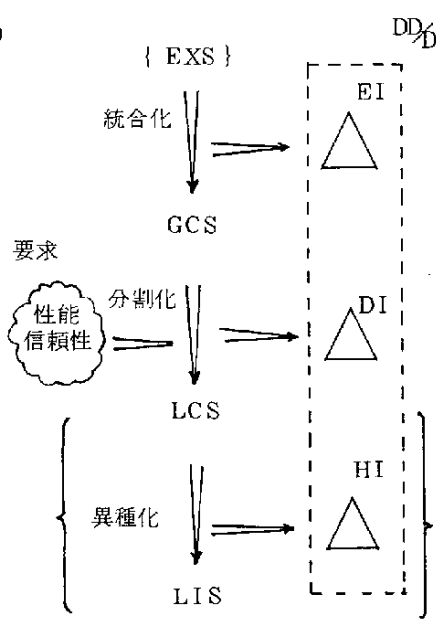


図 3.3 分割型設計

まず各応用毎にEXSが定義され、これ等を統合した1つのGCSが定義される。これを統合化とする。次に各GCSを、応答性、信頼性、データ獲得性、安全性等を考慮して、ある目標に合う様にLCSに分割する。これを分割化とする。最後に、各DBS固有のデータモデルでLCSを表し、LISとする。これを異種化とする。

初期設計として、統合か分割かのどちらかの方法がとられた後、実際のDDBS運用を通して両者の方法が混在していく。

3.1.3 DDBSの運用

3.1.2で述べた様にして設計された四層のDDBSに対して、利用者は自分の応用毎の視野を提供する外部(EX)層で、EXSに対する問合せ/更新演算から成るトランザクション T_E をEX操作言語(EXML)で表し、DDBSに入力する。DDBSは、 T_E を全体概念(GC)

層のGCS上のトランザクション T_G に変換する。この変換は、外部情報EIを用いて行われる。 T_G の参照するデータは、一般に複数のLCS(即ち、異なったDBS)上に存在する。この為、 T_G を、各LCS内のデータ構造に対するトランザクション T_{GL} に変換し、各DBSのローカル処理とDBS間での通信処理によって解を得る。これを分散トランザクション(問合せ/更新)処理とする。 T_{GL} を、各LCS内で独自に処理出来る部分 T_L に分割することをトランザクション分割とする。分散トランザクション処理は、(i) T_G から T_{GL} への変換、(ii) T_{GL} を各LCSの T_L への分割(トランザクション分割)、(iii)DBS間での通信処理から成り、全ての処理は、DIに基づいて行われる。又、 T_G の原子性とインテグリティを保つ為の同時実行制御、コミットメント制御、障害管理が必要になる。各DBSに分割されたトランザクション T_L は、共通モデルに基づいているので、各DBSで実行可能なローカル内部(LI)操作言語(LIML)によるトランザクション T_L に、異種性情報HIを用いて変換され、DBS上で実行され、DDBSで共通なLCデータ構造に基づいた解が得られる。これを、トランザクション(問合せ/更新)変換とする。図3.4に以上のDDBSのトランザクション処理の概要を示す。

3.1.4 共通モデル

DDBSの共通モデルとして何を選ぶかは、DDBSの設計と運用上重要な問題である。異種DBSから成るDDBSでは、共通モデルは、CODASYL、IMS、関係モデルの論理的なデータ構造を表せる必要がある。我々は、種々のデータ構造を表現し得、かつ操作性に優れた関係型のモデルを共通モデルとして採用している。又、共通操作言語としては述語論理に基づき、QUEL[HELDG75]に類似したRQLを用いる。共通(概念)モデルについては、第Ⅱ部を参照されたい。

3.2 DDBSのシステム構成

四層構造に基づいたDDBSのシステム構成について論じる。DDBSは n (≥ 1)個のデータモジュール $DM_1, \dots, DM_n$ が、通信網(CN)で結合されたシステム構成をとる[図3.5]。通信網は、データモジュール間の高信頼な種々の通信(1対1、1対多、多対多)をサポートする網で、広域パケット交換網、ローカルエリア網(LAN)、バス、無線網等がこの例である。

各データモジュール DM_i は、次の6つのモジュール、全体データベースプロセッサ(GDP_i)、ローカルデータベースプロセッサ(LDP_i)、データベースシステム(DBS_i)、異種性情報(H_i)、分散性情報(DI)、関係作業領域(RWS_i)から成る。データベースシステム DBS_i は、既存のDBMSによって管理されたDBSであり、ローカル内部(LI)スキーマ(LIS)に対してLIS操作言語(LIML)で書かれたトランザクションを入力して、これを実行し、結果(問合せならば導出データ)を導出出来るシステムである。ローカルデータベースプロセッサ LDP_i は、既存の DBS_i 上にあって、利用者にローカル概念(LC)層を提供するインタフェースである。 LDP_i は、スキーマ変換(同種化($LIS \rightarrow LCS$)/異種化($LCS \rightarrow LIS$))機能と、トランザクション変換($LCML \rightarrow LIML$)機能とを有している。前者では、LISと

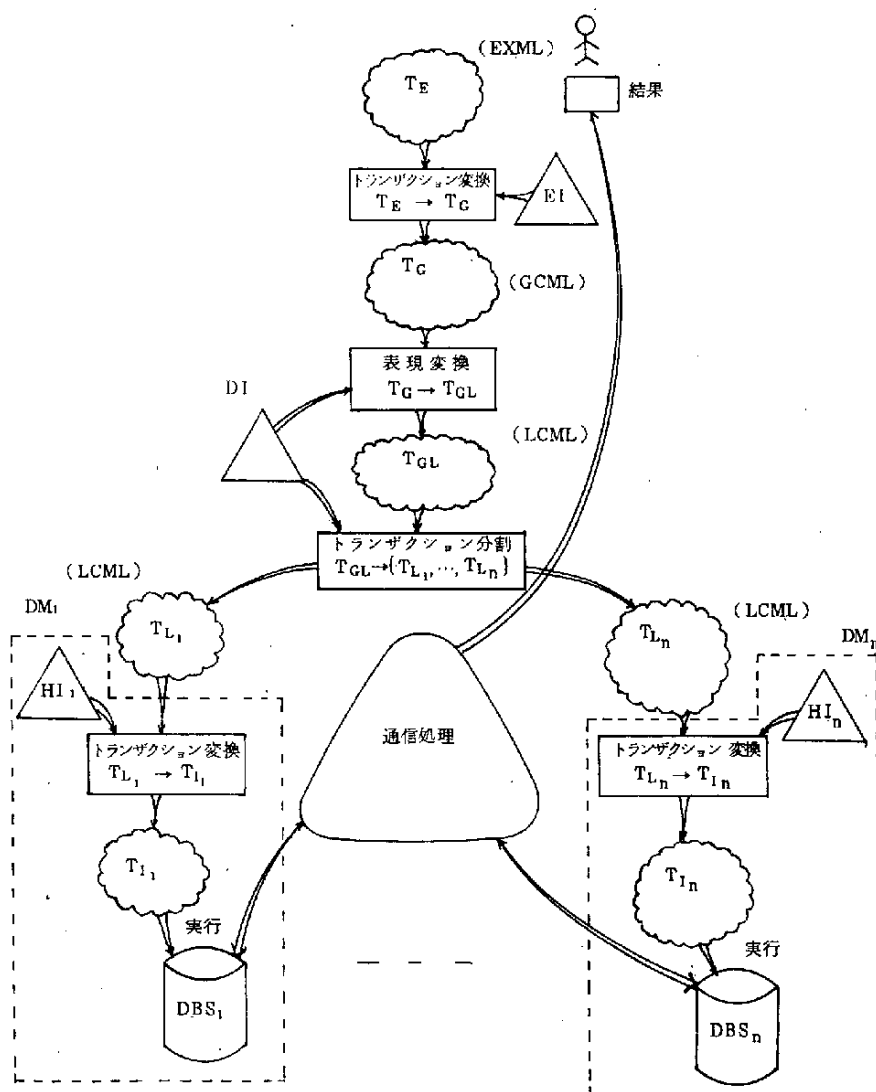


図 3.4 トランザクション処理

LCSとの対応情報を異種性情報 (H I i) として生成し、後者では、この H I i を用いてトランザクションの変換が行われる。トランザクション変換は、LCML トランザクション T_L を、LIML トランザクション T_I に変換し、これを DBS_i で実行させ、検索では結果を共通な関係データ構造として導出し、関係作業領域 RWS_i に関係として格納する。J DDBS の LDP_i は、(i) CODASYL 型の DBS_i の CODASYL スキーマ (LIS) から関係モデルの LC スキーマ (LCS) の生成と、(ii) LCS 上の RQL 問合せ (検索、更新) (LCML) を、COBOL DML (LIML) に変換し、実行し、解を関係として RWS_i に格納する。

全体データベースプロセッサ GDP_i は、 LDP_i 上にあつて、利用者に DDBS 全体概念 (GC) 層を提供するプロセッサである。 GDP_i の機能としては、統合型設計における統合化 ($\{LCS\} \rightarrow GCS$) と分割型設計における分割化 ($GCS \rightarrow \{LCS\}$)、及び分散トランザクシ

ン処理の機能を有している。統合型設計では、DDBSの全体管理者(GA)の定義したGCS定義から分散情報(DI)を生成し、DDBSの全てのDM_jにその完全コピーを送る。分割型設計でも、DIを生成し、他のDM_jに送る。分散トランザクション処理では、以下を行う。

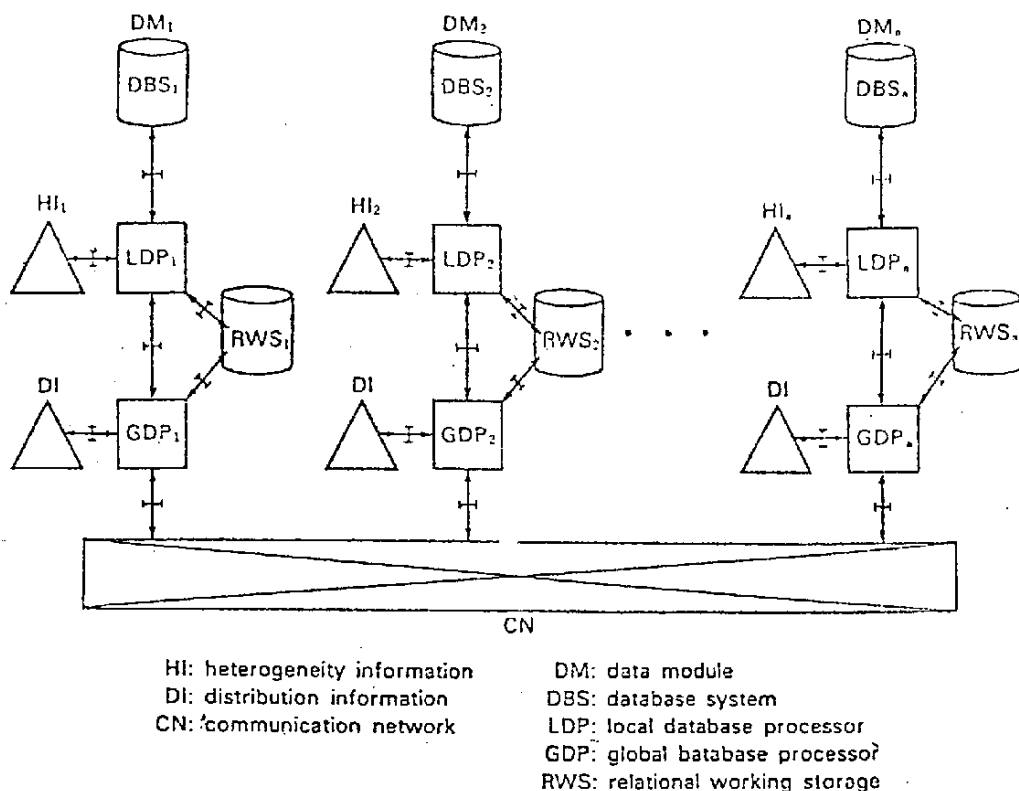


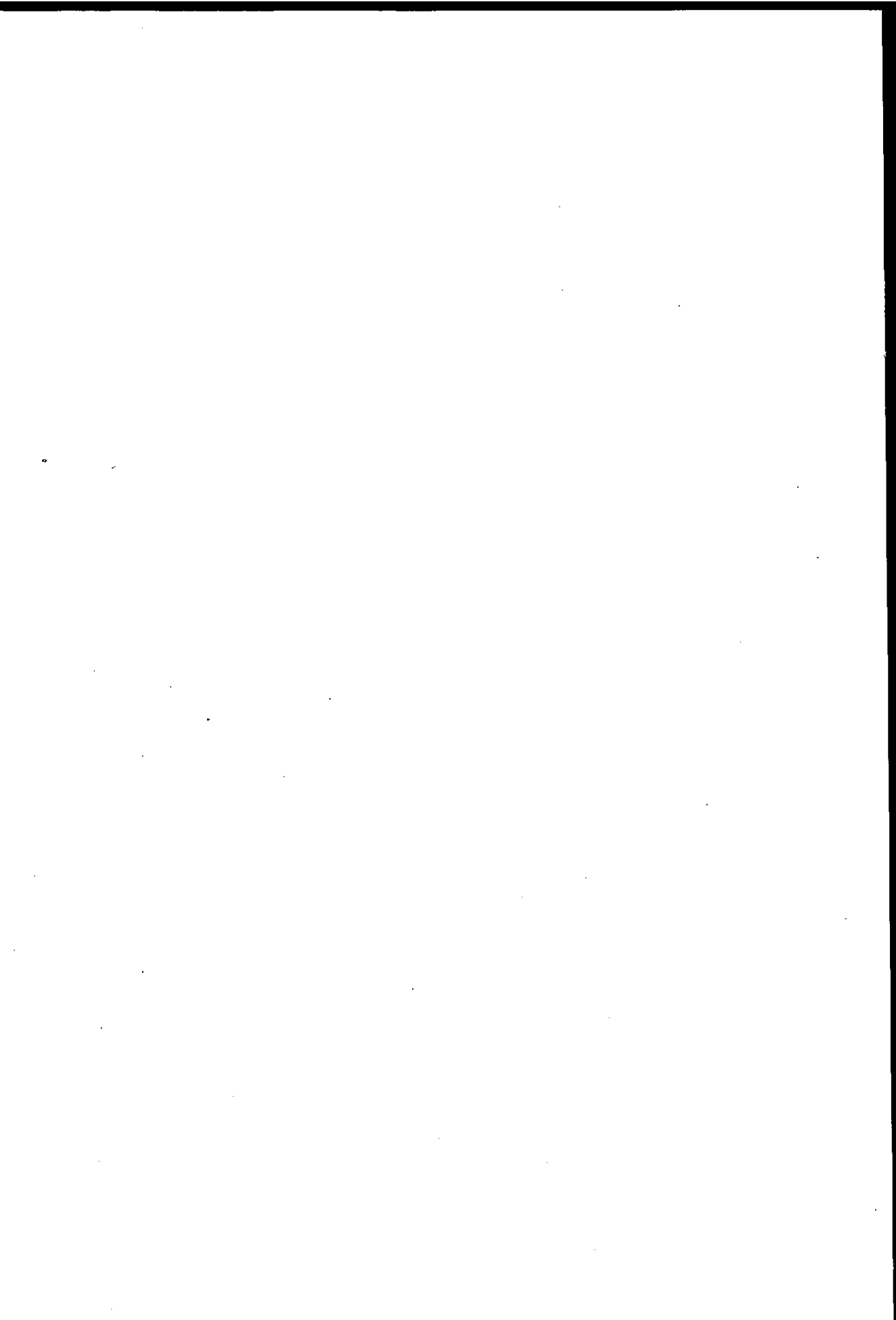
図 3.5 J DDBSアーキテクチャ

- (i) GCMLトランザクション T_G をLCMLで表す。これを T_{GL} とする〔表現変換〕。
- (ii) T_{GL} を、各DM_iで独立して処理出来るトランザクション T_{L_i} を実行し、解をRWS_iに格納する。
- (iii) T_{L_i} を処理したGDP_iは、RWS_i内の関係上に、互いに通信を行い、DM間での処理(例、DM間の関係の結合)を行う。

分散トランザクション処理は、DIに基づいて行われる。各DM_iは、同一情報から成るDIを有している。J DDBSにおけるGDP_iは以下の様な機能を持つ。

- (1) LC関係上の視野としてのGCS関係定義式より分散情報DIを生成する。
- (2) RQLによりGC問合せを分散処理して解を利用者に渡す。

DDBSにおける管理システム(DDBMS)は、各DM_iのLDP_i、GDP_iであり、DD/Dは、HI_iとDI(及びEI)の集合である。以降J DDBS〔TAKIM78-82〕におけるLDPとGDPについて述べる。



第Ⅱ部 異種性問題

第Ⅱ部 異 種 性 問 題

本部では、分散型データベースシステム (DDBS) が、種々のデータベースシステム (DBS) を含むことによって生じる異種性問題の解法について論じる。

まず第1章では、DBSの異種性の定義を行う。第2章では、DDBSの共通モデルとして各データモデルを概念化した概念モデルを定義する。第3章では、CODASYLモデルを4つの階層に分離して、各層のデータ構造と操作演算とを定義する。第4章では、概念モデルを特性化したローカル概念 (LC) モデルを定め、これと概念的なCODASYLモデルとの等価性を示す。第5章では、LCデータ構造上の関係問合せ (RQL) を、論理CODASYLモデルの非手続的な問合せ言語 (CQL) で表せることを示す。第6章と7章では、この非手続的な問合せを、CODASYLモデルのデータ操作言語 (DML) で表せることを示すとともに、出力ファイル数と操作を最少化するDML生成アルゴリズムとその正しさを示す。第8章では、CODASYL DBS上に、異種性問題を解決する為に実現した関係インタフェースシステムとしてのローカルデータベースプロセッサ (LDP) について簡単に触れる〔詳細は第Ⅳ部〕。第9章では、LDPの性能面、利用面からの評価結果を述べる。

1. データベースシステム (DBS) の異種性

データベースシステム (DBS) とは、あるデータ構造と、このデータ構造上の操作演算とを利用者に提供するオートマトンと考えられる。データモデルとは、データ構造と操作演算とを意味するとする。データモデルとしては、関係モデル [CODDE70]、網 (CODASYL) モデル [CODAS73, OLLET78]、階層 (IMS) モデル [DATEC76] の3つが代表的であり、現在の商用DBSで用いられている。よって、ここではDBSの異種性を、各データモデルの相違と定義する。DBSの相違点としては、他に次の様なものがある。

- (i) ハードウェア
- (ii) OS
- (iii) インテグリティ、セキュリティの管理機能 (e.g., ロックの単位)
- (iv) DBMSへのログオン手順
- (v) データ値の表現 (単位等)

JDDBSでは、(i)~(v)は、通信網 (CN) 等が解決するものとして、DBSを考えるうえで、最も本質的な相違点であるデータモデルについて検討する。

〔定義〕

データベースシステム (DBS) の異種性 (heterogeneity) とは、各データモデルの相違とする。□

2. 概念モデル

2.1 データモデルの概念化

データベースシステム(DBS)は、利用者に対して、あるデータモデルに基づいたデータ構造と、この上の演算を提供するオートマトンと考えられる。データモデルとしては、関係[CODDE70]、階層[DATEC77]、網[CODAS73]の3つが著名である。本報告書では、DBSの異種性を各データモデルの相違とし、データモデルはデータ構造とこれ上の操作演算で定まるとする。異種DBSの統合を行うためには、まず各DBS上にDDBS全体の共通モデルに基づいた視野を設ける必要がある。この共通モデルを概念モデルと呼ぶ。概念モデルは、各モデルの(更新演算にする制約を含めた)データ構造を正確に表せる必要がある。

階層、網モデルは、実体と関連との2種概念を有していて、更新に対する制約は関連の型として表される。例えば、網型のCODASYLモデル[CODAS73, OLLET78][3章]では、レコード型は実体、セット型は実体間の1対 m (≥ 0)関係を表している。更新に対する制約としてのメンバシップクラスはセット型の型であり、型は更新演算に異った意味(更新の実行条件、更新の範囲)を与える。階層型のIMSモデル[DATEC76]では、セグメント型が実体、階層パスが実体間の1対 m 関係を表している。以上より、概念モデルは、実体と関連との2つの独立な概念を有している必要がある。各データモデルのデータ構造は、概念モデルの実体間の関連の種類(型)により特性化して表せる。本報告書では、特にCODASYLモデルを取りあげ、概念モデルの特性化によってこれを表す問題を論じる。

2.2 概念データ構造

概念データ構造 $\mathcal{A}$ は、時間不変な概念スキーマ $\mathcal{A}$ と、 $\mathcal{A}$ に実際に値を与え時間可変な概念データベース $\mathcal{B}$ とから成る($\mathcal{A}=(\mathcal{A}, \mathcal{B})$)。概念スキーマ $\mathcal{A}$ は、2種の関係スキーム、E関係スキームとR関係スキーム、とから成る。

[定義]

E関係スキーム E は、属性集合 $\mathcal{Q}_E = \{\textcircled{E}, a_1, \dots, a_m\}$ ($m \geq 0$)とインテグリティ条件 Γ_E とから成る($E = (\mathcal{Q}_E, \Gamma_E)$)。各属性 $a \in \mathcal{Q}_E$ に対して、 $\text{dom}(a)$ は a の可能な値の集合としての定義域を表す。 $\mathcal{Q}_E$ の中で、属性 $\textcircled{E}$ は主属性と呼ばれ、 $\text{dom}(\textcircled{E})$ は、概念データベース内で一意な値(識別子)の集合である。又、 $\text{dom}(\mathcal{Q}_E) = \text{dom}(\textcircled{E}) \times \text{dom}(a_1) \times \dots \times \text{dom}(a_m)$ とする。インテグリティ条件 Γ_E は、 $\mathcal{Q}_E$ が取り得る値の組(これをE組とする)を定める意味的な述語である。E関係スキーム E に実際に値を与えたE組の集合を、E関係 E とする。即ち、

$$E \triangleq \{ e \mid e \in \text{dom}(\mathcal{Q}_E) \wedge \Gamma_E(e) \}$$

E組 $e \in E$ の属性 $\alpha \in \mathcal{Q}_E$ の値を $\alpha(e)$ とする。属性の並び $\alpha_1, \dots, \alpha_m$ ($\alpha_i \in \mathcal{Q}_E$) の値は、 $\alpha_1(e), \dots, \alpha_m(e)$ を連結したもので、 $\alpha_1 \dots \alpha_m(e)$ と表す。E関係Eの中で、主属性 $\otimes E$ の値は一意であるので、 $\otimes E$ は主キーとなる。ここで、E関係スキームを Γ_E を省略して $\underline{E}(\otimes E, \alpha_1, \dots, \alpha_m)$ ($m \geq 0$) と表す。下線を引いた属性は、主キーを表すものとする。□

〔定義〕

任意の n (≥ 2) 個のE関係スキーム $\underline{E}_1, \dots, \underline{E}_n$ を考える(但し、各 $\underline{E}_i$ は、互いに異っている必要はない)。この時、ある属性集合 $\mathcal{Q}_R'$ ($\mathcal{Q}_R' \cap \mathcal{Q}_{E_i} = \emptyset$) があり、ある関係: $\text{dom}(\mathcal{Q}_{E_1}) \times \dots \times \text{dom}(\mathcal{Q}_{E_m}) \times \text{dom}(\mathcal{Q}_R')$ が存在するときに、これをR関係スキーム $\underline{R} = (\underline{E}_1, \dots, \underline{E}_n, \mathcal{Q}_R', \Gamma_R)$ で表す。 Γ_R は、インテグリティ条件で、関係に含まれるべき組を定める述語である。 $\underline{R}$ に実際に値を与えたものをR関係Rとする。即ち

$$R \triangleq \{ r \mid r \in \prod_{i=1}^n \text{dom}(\mathcal{Q}_{E_i}) \times \text{dom}(\mathcal{Q}_R') \wedge \Gamma_R(r) \}.$$

各要素 $r \in R$ を、R組とする。

$\mathcal{Q}_R' = \{ \alpha_1, \dots, \alpha_m \}$ ($m \geq 0$) とする。各E関係 $\underline{E}_i$ において、 $\mathcal{Q}_{E_i}$ 内の主属性 $\otimes E_i$ は主キーであるので、 $\underline{R}$ 内の $\mathcal{Q}_{E_i}$ を $\otimes E_i$ で表すことにする。このとき、R関係スキームを、 Γ_R を省略して、 $\underline{R}(\otimes E_1, \dots, \otimes E_n, \alpha_1, \dots, \alpha_m)$ と記す。R関係Rは次の様になる。

$$R \triangleq \{ r' \mid r' = \otimes E_1 \dots \otimes E_n \alpha_1 \dots \alpha_m(r) \wedge r \in \prod_{i=1}^n \text{dom}(\mathcal{Q}_{E_i}) \times \prod_{j=1}^m \text{dom}(\alpha_j) \wedge \Gamma_R(r) \}$$

Γ_R の表す1つの条件として、次のものがある。即ち、常に、全てのR組 $r \in R$ に対して、全ての $\underline{E}_i$ のE関係 $\underline{E}_i$ 内に、 $\otimes E_i(r) = \otimes E_i(e_i)$ なるE組 e_i が存在せねばならない。この時、 r に対する $e_1 \in E_1, \dots, e_n \in E_n$ (ここで、 $\otimes E_i(e_i) = \otimes E_i(r)$) を、 r によって関係づけられているという。□

概念スキーマ $\mathcal{B}$ 内の各EとR関係スキームに実際に値を与えたEとR関係の集合を、概念データベース $\mathcal{B}$ とする。単に、関係又は関係スキームというときは、E又はR関係スキーム又は関係を表すとする。

2.3 概念操作言語 (CML)

概念データベース $\mathcal{B}$ 内の関係の操作はQUEL [HELDG 75] と同様な構文を持つ概念操作言

語 (CML) によって行える。

2.3.1 検 索

$X_1, \dots, X_L$ を、 $\mathcal{B}$ 内の 1 個の任意の関係スキームとし、 $\underline{T}(a_1, \dots, a_k)$ を利用者の必要とする目標関係スキームとする。検索とは、 $\mathcal{Q}_{X_1}, \dots, \mathcal{Q}_{X_L}$ から $\mathcal{Q}_T$ を定める検索関数 $f(\mathcal{Q}_{X_1}, \dots, \mathcal{Q}_{X_L}) = \mathcal{Q}_T$ を定義することである。この f を $X = X_1 \times \dots \times X_L$ に作用させる目標関係 R が得られる。

概念操作言語 (CML) では、 $\mathcal{B}$ 内の関係 X_i の各組は組変数 x_i を用いて特定される。組変数は (1) の様に定義される。

$$\text{range } (x_1, X_1) \dots (x_L, X_L); \quad \dots\dots\dots (1)$$

$$\text{retrieve into } T(t1) \text{ where } \text{qual} \quad \dots\dots\dots (2)$$

検索関数 f は、(2) 式の様に記述される。 T は目標関係名である。 $t1$ は目標リストと呼ばれ、 $t1 ::= a_1 = \text{exp}_1, \dots, a_k = \text{exp}_k$ であり、各 exp_i は属性又は属性上の算術式 (e.g. $x_k \cdot a_k + x_j \cdot a_j$) である。 $w \in X (= X_1 \times \dots \times X_L)$ に対して、 $t1(w)$ は組 $(a_1, \dots, a_m)$ を与える関数 ($t1: X \xrightarrow{1} \text{dom}(\mathcal{Q}_{R_i}) \rightarrow \text{dom}(\mathcal{Q}_T)$) と考えられる。 qual は、結合述語 $\text{jf}_{ij} ::= x_i \cdot a_i \theta x_j \cdot a_j$ ($\text{dom}(a_i) = \text{dom}(a_j)$) と制限述語 $\text{rf}_i ::= x_i \cdot a_i \theta v$ (v は定数) とから成る論理式である ($\theta \in \{<, <=, =, \geq, >, \neq\}$)。結合述語は、 $\text{jf}_{ij}: \text{dom}(\mathcal{Q}_{R_i}) \times \text{dom}(\mathcal{Q}_{R_j}) \rightarrow \{\text{true}, \text{false}\}$ 、制限述語は $\text{rf}_i: \text{dom}(\mathcal{Q}_{R_i}) \rightarrow \{\text{true}, \text{false}\}$ なる述語である。ここで、 x_k は関係 X_k の組変数であり、 $x_k \cdot a_k$ は X_k の属性を表す ($k = i, j$)。 $w \in X$ に対して、 $\text{qual}(w)$ は true 又は false を与える論理式と考えられる。 ($\text{qual}: X \xrightarrow{1} \text{dom}(\mathcal{Q}_{R_i}) \rightarrow \{\text{true}, \text{false}\}$) 即ち、 $R = \{r \mid r = (a_1, \dots, a_k) = t1(w) \wedge \text{qual}(w) \wedge w \in X_1 \times \dots \times X_L\}$ 。又、 R は冗長な組を持たない。概念モデルの検索では、主属性が識別子を取る事から、主属性間の比較演算子として $=$ と $\neq$ のみが許され、更に主属性上の算術演算は許されない。*)

2.3.2 基本更新演算

$E(\oplus E, a_1, \dots)$ と $R(\oplus E_1, \dots, \oplus E_n, b_1, \dots)$ を各々、 $\mathcal{B}$ 内の任意の E と R 関係とする。この 2 種の関係に対して、消去、追加、置換があるので、次の 6 種の基本演算がある。

- (1) $\text{del}(e, E \mid e \in E)$; 組 $e \in E$ を除く。同時に、 $\oplus E$ を主属性として持つ全ての R 関係 R から $\oplus E(e) = \oplus E(r)$ なる組 r を除く。
- (2) $\text{del}(r, R \mid r \in R)$; 組 $r \in R$ を除く。
- (3) $\text{app}(e, E)$; 関係 E に組 e を加える。
- (4) $\text{app}(r, R \mid e_1 \in E_1, \dots, e_n \in E_n)$; 各 E_i が $\oplus E_i(e_i) = \oplus E_i(r)$ なる組 e_i を持つ ($i = 1, \dots, n$) とき、 R に組 r を加える。
- (5) $\text{rep}(e, e', E \mid e \in E)$; 組 $e \in E$ の非主属性値を e' で置換する。
- (6) $\text{rep}(r, r', R \mid r \in R)$; 組 $r \in R$ の非主属性値を r' で置換する。

*) 同様に、集積関数としては、[STONM76] の count と any のみが許される。

3. CODASYLデータベースシステム

本章では、CODASYL DBSの定義を行う。

3.1 CODASYL DBSの階層構造

〔CODAS73〕と〔OLLET78〕に基づいたCODASYL DBSについて考える。
CODASYL委員会では、〔CODAS78〕等の新提案を出しているが、実際の商用DBS（例、AIM（Facom）、ADBS（NEC））では、〔CODAS73〕に近いものがようやく実現されているのが現状である。このため、〔CODAS73〕について検討することにする。

CODASYL DBSは、図3.1に示す様な4つの階層から構成されると考えられる。

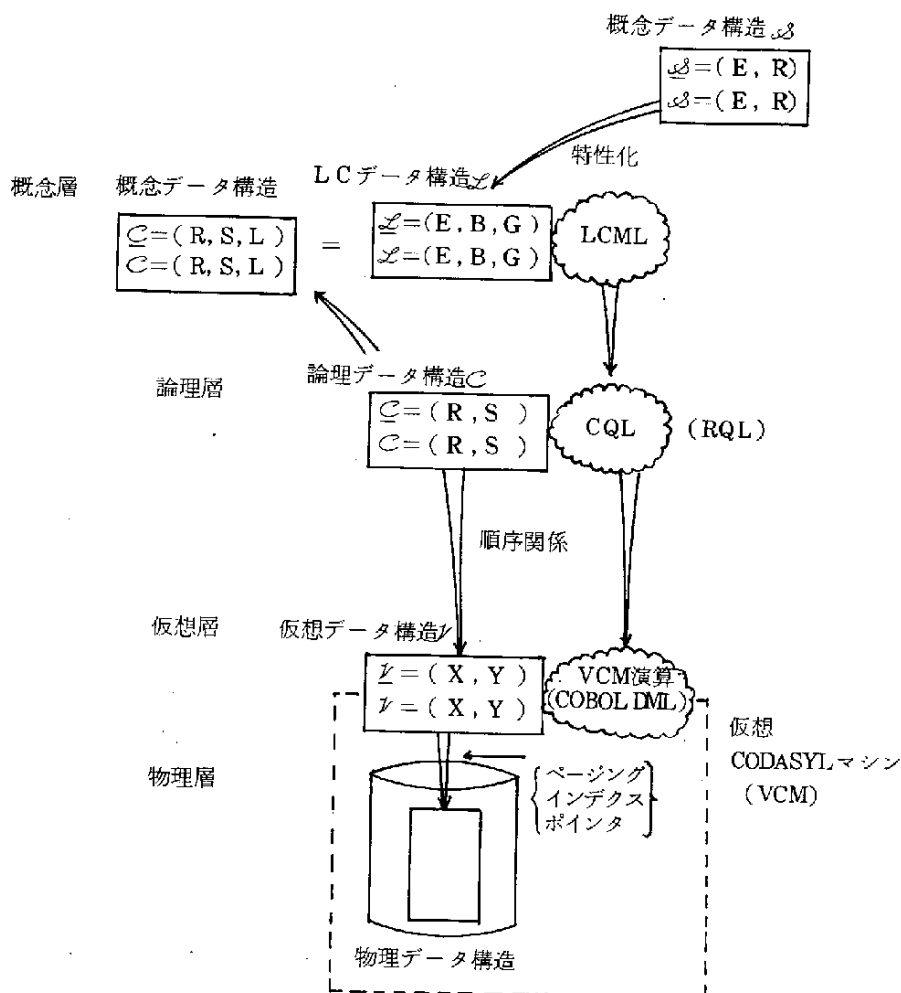


図3.1 CODASYL DBSの階層化

3.2 論理CODASYLモデル

本節では、概念層と仮想層の中間に位置づけられる論理層のデータ構造と、操作演算を定義する。

3.2.1 論理データ構造

CODASYL DBSの論理層の論理データ構造 C は、時間不変な論理スキーマ $\underline{C}$ と、 $\underline{C}$ に実際に値を与えた、時間可変な論理データベース C とから成る($C=(\underline{C}, C)$)。論理スキーマ $\underline{C}$ は、レコード型(又はR型)の集合 $\underline{R}$ と、セット型(又はS型)の集合 $\underline{S}$ とからなる($\underline{C}=(\underline{R}, \underline{S})$)。各レコード型 $\underline{R}$ は、項目集合 $\underline{Q}_R=\{\textcircled{R}, t_1, \dots, t_m\}$ ($m \geq 0$)と、インテグリティ条件 Γ_R とから成る($\underline{R}=(\underline{Q}_R, \Gamma_R)$)。各項目 $\tau \in \underline{Q}_R$ に対して、 $\text{dom}(\tau)$ は、 τ の取り得る値の集合を表し、 τ の定義域とする。項目 $\textcircled{R}$ は、特にデータベースキー(dbキー)と呼ばれ、 $\text{dom}(\textcircled{R})$ は識別子の集合である。 t_i は、データ項目である($i=1, \dots, m$)。 $\pi_R \triangleq \{t_1, \dots, t_m\} \triangleq$ データ項目の集合とする。インテグリティ条件 Γ_R は、 $\underline{Q}_R$ 内の各項目値の組を、意味的に定める述語である。 Γ_R を満足するこの組の集合を、レコード実現値集合(又は、RO集合) R とする。即ち

$$R \triangleq \{r \mid r \in \text{dom}(\underline{Q}_R) \wedge \Gamma_R(r)\}$$

ここで、 $\text{dom}(\underline{Q}_R) \triangleq \bigcup_{\tau \in \underline{Q}_R} \text{dom}(\tau)$ である。

R 内の各要素 $r \in R$ を、レコード実現値(又は、 R 実現値)とする。各項目 $\tau \in \underline{Q}_R$ のレコード実現値)とする。各項目 $\tau \in \underline{Q}_R$ のレコード実現値 $r \in R$ の値を、 $\tau(r)$ と表す。 $r \in R$ の項目集合 $\{t_1, \dots, t_m\} \subseteq \underline{Q}_R$ の値を、 $t_1, \dots, t_m(r)$ と表す。全ての $r \in R$ に対して、dbキー $\textcircled{R}$ の値は一意であるので、 $\textcircled{R}$ は $\underline{R}$ の主キーとなる。又、任意のレコード実現値(RO)集合 R と R' ($R \neq R'$)において、 $\forall r \in R \quad \forall r' \in R' \quad \textcircled{R}(r) \neq \textcircled{R}(r')$ 。

よって、 $\textcircled{R}(r)$ を $\delta(r)$ と表す。又、dbキー値を、利用者は見れないので、 $\textcircled{R}$ を除いてレコード型を $\underline{R}(t_1, \dots, t_m)$ (Γ_R も省略する)と表す。 R に対するレコード実現値(RO)集合の集合を R とする。

論理データ構造 C では、レコード実現値(RO)集合間の関係として部分関数を表せる。ある2つのRO集合 R_2 から R_1 にある部分関数 f が存在するとき、 C では、これをセット型(又はS型)で表す。セット型 $\underline{S}$ は、部分関数 f の定義域のRO集合のレコード型 $\underline{R}_1$ と値域のレコード型 $\underline{R}_2$ 、及びインテグリティ条件 Γ_S とから成る($\underline{S}=(\underline{R}_1, \underline{R}_2, \Gamma_S)$)。ここで、 $\underline{R}_1$ を $\underline{S}$ の親レコード型、 $\underline{R}_2$ を $\underline{S}$ の子レコード型と呼び、図3.2の様に $\underline{R}_1$ から $\underline{R}_2$ への有向辺で図示される。インテグリティ条件 Γ_S は、 $\underline{Q}_{R_1}$ と $\underline{Q}_{R_2}$ とから意味的に導出される述語であり、セット型 $\underline{S}$ として取り得る R_1 と R_2 のレコード実現値の対を定める。 $\underline{S}$ に実際に値を与えたものを、セット実現値集合(又はSO集合) S とする。

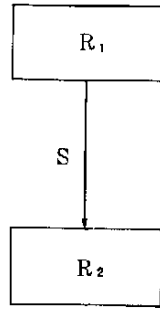


図 3.2 セット型 $\underline{S} = (R_1, R_2)$

$$\underline{S} \triangleq \{ s \mid s \in R_1 \times R_2 \wedge I_S(s) \}$$

各 $s = (r_1, r_2) \in \underline{S}$ を、セット実現値 (又は $\underline{S}$ 実現値) とし、 r_1 を r_2 の親、 r_2 を r_1 の子レコード実現値とする。セット型 $\underline{S}$ は、 R_2 から R_1 への部分関数を表すので、各 $r_1 \in R_1$ の $\underline{S}$ の子 $r_2 \in R_2$ は、複数 (0 以上) 存在する。一方、各 $r_2 \in R_2$ に対する $\underline{S}$ の親 $r_1 \in R_1$ は、高々 1 つ存在するだけである。ここで、次を定義する。

各 $r_1 \in R_1$ に対して、

$$\begin{aligned} \underline{\text{member}}(r_1) &\triangleq \{ r_2 \in R_2 \mid (r_1, r_2) \in \underline{S} \} \\ &\triangleq r_1 \text{ の子の集合。} \end{aligned}$$

各 $r_2 \in R_2$ に対して、

$$\begin{aligned} \underline{\text{owner}}(r_2) &\triangleq \{ r_1 \in R_1 \mid (r_1, r_2) \in \underline{S} \} \\ &\triangleq r_2 \text{ の親 } r_1 \text{。} \end{aligned}$$

セット型 $\underline{S}$ において、各 $r_1 \in R_1$ の子レコード実現値集合 $\underline{\text{member}}(r_1)$ 内の各レコード実現値 r_2 のあるデータ項目 $t \in \pi_{R_2}$ の値が、 $\underline{\text{member}}(r_1)$ 内で一意であるとき、これはインテグリティ条件 I_S の 1 つとなる。このデータ項目 $t \in \pi_{R_2}$ を、セット型 $\underline{S}$ の DNA (duplicates are not allowed) 項目と呼ぶ。即ち、

$$\begin{aligned} &\exists t \in \pi_{R_2} \\ &\forall r_1 \in R_1 \forall r_2, r_2' \in \underline{\text{member}}(r_1) (r_2 \neq r_2') \\ &\quad t(r_2) \neq t(r_2') \end{aligned}$$

レコード型 R_1 と R_2 において、各々 db キー $\textcircled{R}_1$ と $\textcircled{R}_2$ は主キーとなるので、セット実現値 $s = (r_1, r_2) \in \underline{S}$ を、レコード実現値 r_1 と r_2 の db キー値の対 $(\delta(r_1), \delta(r_2))$ としても表せる。よって、セット実現値集合 $\underline{S}$ を、次の様にする。

$$S \triangleq \{ s' \mid s' = (\delta(r_1), \delta(r_2)) \wedge s = (r_1, r_2) \in R_1 \times R_2 \wedge \Gamma_S(s) \}$$

Sに対するセット実現値(SO)集合の集合をSとする。

論理スキーマ $\underline{C} = (\underline{R}, \underline{S})$ に対する論理データベースCはRO集合の集合Rと、SO集合の集合Sとから成る($C = (R, S)$)。

論理スキーマ内の各セット型 $\underline{S} = (\underline{R}_1, \underline{R}_2, \Gamma_S)$ に、論理データベースを変化させる更新演算に対する制約としての型(色)を定義出来る。検索利用だけの場合は、型は不要である。 $\text{stype}(\underline{S})$ は、セット型 $\underline{S}$ の型を表すとする。型としては、M(andatory)/A(utomatic)とO(ptional)/M(annual)との2種がある。*)

$$\text{stype}(\underline{S}) \in \{M/A, O/M\}$$

セット型 $\underline{S}$ の型の意味については、3.2.2で述べる。

型づけされたセット型から成る論理スキーマ $\underline{C}$ は、次の制約を持つ。

[スキーマ制約]

スキーマ $\underline{C}$ は、全てのセット型 $\underline{S}$ の型がM/A($\text{stype}(\underline{S}) = M/A$)なる有向閉路を含んで
はならない。□

図3.3は、従業員(EMP)、プロジェクト(PROJ)、論文(REP)、従業員の家族(FAMILY)を表す論理スキーマの例を示している。箱はレコード型を、実線と点線の矢は各々、M/AとO/M型のセット型を表している。

3.2.2 論理操作演算—検索

前項(3.2.1)で定義した論理データ構造上の論理操作演算を定義する。まず、検索について考える。 $\underline{R}_1, \dots, \underline{R}_h$ を論理スキーマ内の任意の $h (\geq 1)$ 個のレコード型とする(互いに異っている必要はない)。 $\underline{R} = \{\underline{R}_1, \dots, \underline{R}_h\}$ とする。 $\underline{S}_1, \dots, \underline{S}_p$ (ここで、 $\underline{S}_i = (\underline{R}_{i_1}, \underline{R}_{i_2}, \Gamma_{S_i})$ 、 $\underline{R}_{i_1}, \underline{R}_{i_2} \in \underline{R}$)を、 $\underline{C}$ 内の任意の p 個セット型(互いに異っている必要はない)とする。利用者の必要とする目標集合Tは、属性集合 $\mathcal{Q}_T = \{a_1, \dots, a_k\} (k \geq 1)$ から成るとする。論理検索演算とは $f(\mathcal{Q}_{R_1}, \dots, \mathcal{Q}_{R_h}, \underline{S}_1, \dots, \underline{S}_p) = \mathcal{Q}_T$ なる検索関数 f を定義することである。 f を直積 $R_1 \times \dots \times R_h \times S_1 \times \dots \times S_p$ に作用させると、 $T \subseteq \text{dom}(\mathcal{Q}_T)$ が得られる。

*) S型 $\underline{S}$ のメンバシップクラスは[CODAS73]では消去演算に対する除去クラス $\text{rc}(\underline{S})$ と、追加演算に対する格納クラス $\text{sc}(\underline{S})$ とがある。 $\text{rc}(\underline{S}) \in \{M(\text{andatory}), O(\text{ptional})\}$ 、 $\text{sc}(\underline{S}) \in \{A(\text{utomatic}), M(\text{annual})\}$ であり、 $\text{mc}(\underline{S}) = \text{rc}(\underline{S}) / \text{sc}(\underline{S}) \in \{M/A, M/M, O/M, O/A\}$ である。

しかし、一般には $\text{mc}(\underline{S}) \in \{M/A, O/M\}$ だけが用いられる。

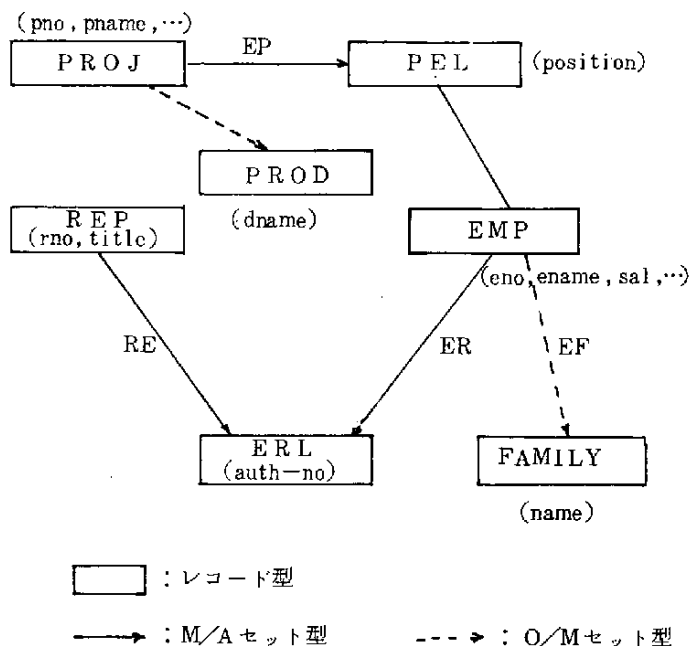


図 3.3 論理スキーマ

A CODASYL 問合せ言語 (CQL)

論理データベース上の検索言語として、述語論理に基づいた CODASYL 問合せ言語 [TAKIM 80、82] を定義する。CQL によって検索演算は、次の様に書かれる。

$\text{var } (r_1, R_1) \cdots (r_h, R_h);$
 $\text{get into } T(t_1) \text{ where qual};$ } (1)

R_i は、論理スキーマ C 内のレコード型名である ($i = 1, \dots, h$)。各 R_i は、互いに異なっている必要はない。 r_i は、論理データベース C 内のレコード実現値 (RO) 集合 R_i の実現値を値としてとる実現値 (O) 変数とする。 var 文は、 R_i に対する O 変数 r_i を定義する。

T は、目標集合名であり、 $T = (Q_T)$ ($Q_T = \{a_1, \dots, a_k\}$ ($k \geq 1$)) であり各 a_i を目標属性とする。(1) 式内で、項目 $t \in Q_{R_i}$ は、 $r_i.t$ と表される。 t_1 は、目標リストと呼ばれ、次の様な構文からなる。

$t_1 ::= a_1 = \text{exp}_1, \dots, a_k = \text{exp}_k$ (2)
 $\text{exp}_i ::= v \mid r.t \mid \text{項目上の算術式}$

ここで、 v は定数である。式 exp_i は、目標式とする。式 exp_i に対して、 $\text{var}(\text{exp}_i)$ は exp_i の参照する O 変数の集合とする。 $\text{var}(\text{exp}_i) = \{r_{i_1}, \dots, r_{i_l}\}$ のとき、 exp_i は関数 $\text{exp}_i :$

$\prod_{j=1}^I \text{dom}(\mathcal{Q}_{R_{ij}}) \rightarrow \text{dom}(a_i)$ である (r_{ij} は R 型 R_{ij} の O 変数とす) である。よって目標リスト tl は、関数 $tl: \text{dom}(\mathcal{Q}_{R_1}) \times \dots \times \text{dom}(\mathcal{Q}_{R_k}) \rightarrow \text{dom}(\mathcal{Q}_T)$ なる関数と考えられる。

$qual$ は、条件式と呼ばれ、次の様に積正規化された論理式である。

$$\begin{aligned} qual &::= rf \wedge sf \wedge jf \\ rf &::= rf_1 \wedge \dots \wedge rf_k \\ rf_i &::= rp_i \mid rp_i \vee rf_i \mid rp_i \wedge rf_i \mid (rf_i) \\ rp_i &::= r_i . t \quad \theta \quad v \\ \theta &::= < \mid \leq \mid = \mid \geq \mid > \mid \neq \end{aligned}$$

ここで、 v は定数であり、 r_i は RO 集合 R_i の実現値 (O) 変数であり、 $t \in \mathcal{Q}_{R_i}$ である。 rf_i は、O 変数 r_i についての制限述語 rp_i から成る論理式である ($rf_i: \text{dom}(\mathcal{Q}_{R_i}) \rightarrow \{\text{true}, \text{false}\}$)。rc は、制限式の積となる。

$$\begin{aligned} sf &::= sp \wedge \dots \wedge sp \\ sp &::= P_S(r_i, r_j) \end{aligned}$$

セット型 $S = (\underline{R}_i, \underline{R}_j, I_S)$ に対して、 r_i と r_j は各々、 $\underline{R}_i$ と $\underline{R}_j$ の O 変数とする。この時、セット述語 $P_S(r_i, r_j)$ ($P_S: \text{dom}(\mathcal{Q}_{R_i}) \times \text{dom}(\mathcal{Q}_{R_j}) \rightarrow \{\text{true}, \text{false}\}$) は、次の意味をもつ。

$$P_S(w_i, w_j) = \begin{cases} \text{true} & \text{if } w_i \in R_i \wedge w_j \in R_j \wedge \\ & (\delta(w_i), \delta(w_j)) \in S \\ \text{false} & \text{otherwise} \end{cases}$$

sf は、セット述語の積である。

$$\begin{aligned} jf &::= jf_{i_1 j_1} \wedge \dots \wedge jf_{i_q j_q} \\ jf_{ij} &::= jp_{ij} \mid jp_{ij} \wedge jf_{ij} \mid jp_{ij} \vee jf_{ij} \mid (jf_{ij}) \\ jp_{ij} &::= r_i . t \quad \theta \quad r_j . u \end{aligned}$$

ここで、 r_i と r_j は、各々 R_i と R_j の O 変数である。又、 $t \in \mathcal{Q}_{R_i}$ 、 $u \in \mathcal{Q}_{R_j}$ である。 $\theta \in \{<, \leq, =, \geq, >, \neq\}$ 、但し、 t と u が db キーのときには、 θ として $=$ と $\neq$ のみが許される。 jf_{ij} は、O 変数 r_i と r_j のみを参照する結合述語から成る論理式である ($jf_{ij}: \text{dom}(\mathcal{Q}_{R_i}) \times \text{dom}(\mathcal{Q}_{R_j}) \rightarrow \{\text{true}, \text{false}\}$)。 jf は、結合式の積である。

以上より、条件式 $qual$ は、論理式 ($qual : \text{dom}(Q_{R_1}) \times \dots \times \text{dom}(Q_{R_k}) \rightarrow \{\text{true}, \text{false}\}$) と考えられる。

(1)式のCQL検索演算の意味は、次の様である。

$$T \triangleq \{ t \mid t = t_1(r_1, \dots, r_k) \wedge_{i=1}^k r_i \in R_i \wedge qual(r_1, \dots, r_k) \}$$

但し、 T は冗長な組を含まないものとする。CQLで(1)式の様に書かれた検索演算を、CODASYL問合せ(CQ)とする。

例として、図3.3の論理スキーマに対する検索要求“鈴木を第1著者とする論文名と、その共著者を求めよ”を考える。これはCQLで次の様に書ける。

```
var(e1, EMP)(e2, EMP)(er1, ERL)(er2, ERL)(r, REP);
get into T(r.title, er2.auth-no, e2.ename) where
e1.ename = "鈴木" and ER(e1, er1) and RE(r, er1) and
er1.auth-no = 1 and RE(r, er2) and ER(e2, er2) and
er2.auth-no > er1.auth-no;
```

これと同じ意味を持つCQGを図3.5に示す。

B CODASYL問合せグラフ(CQG)

Aで定義したCQLによる次の任意のCODASYL問合せQを考える。

```
Q var(r1, R1) ... (rk, Rk);
get into T(a1=exp1, ..., ak=expk)
where
jf and rf and sf;
```

ここで、目標リスト内の目標式 exp_i に対して、 $\text{item}(\text{exp}_i)$ は、 exp_i 内の項目の集合とする。
例えば $a.t + b.v$ に対して、 $\text{item}(a.t + b.v) = \{a.t, b.v\}$ となる。集合 $\bigcup_{i=1}^k \text{item}(\text{exp}_i)$ は、問合せQの目標項目集合であり、この中の各項目を目標項目とする。このとき、問合せQのグラフ表現をCODASYL問合せグラフ(CQG)Gを次の様に定義する。

〔定義〕

CODASYL問合せQに対するCODASYL問合せグラフ(CQG)G=(N, E)は、点の集合Nと、辺の集合Eとから成る。点には、O変数点(又は簡単にO点)、目標式点との2種がある。Qの各O変数 r_i に対して、O変数点 r_i が対応する。目標式 $a_j = \text{exp}_j$ に対して、目標式点 $a_j =$

exp_j が対応する。

辺には、2点間の辺と、1点のみに接する辺とがある。2点間の辺としては、主辺、結合辺、目標辺がある。主辺は0(変数)点間の有向辺で、結合辺は0点間の無向辺である。目標辺は、0点から目標式点への有向辺である。問合せQのセット(S)式sfの積の要素としてセット(S)述語 $P_S(r_i, r_j)$ が存在するとき、0変数点 r_i から r_j への主辺が対応し、主辺にはS型名Sが付けられる。結合式jfの積の要素が、0変数 r_i と r_j 間の結合式 jf_{r_i, r_j} を持つとき、0点 r_i と r_j 間に結合辺が対応して存在し、結合式 jf_{r_i, r_j} が付けられる。制限式rf内の積の要素としての0変数 r_i に接する制限辺が対応し、制限式 rf_{r_i} が付される。目標式 $a_i = \text{exp}_i$ (ここで、 $\text{var}(\text{exp}_i) = \{r_{i1}, \dots, r_{iL_i}\} (L_i \geq 0)$)に対して、各0点 $r_{ij} \in \text{var}(\text{exp}_i)$ から、目標点 $a_i = \text{exp}_i$ への目標辺が対応し、 r_{ij} についての目標項目集合 $\text{tl}_{r_{ij}} = \{r_{ij}, t \in \text{item}(\text{exp}_i)\}$ が付される。□

〔定義〕

目標辺に接する0点を、目標点とする。結合辺に接する点を結合点とする。CQG G内の目標点と結合点の和を出力点集合Oとし、Oの各要素をGの出力点とする。□

〔命題3.1〕

全てのCODASYL問合せと、CODASYL問合せグラフとは1対1対応する。

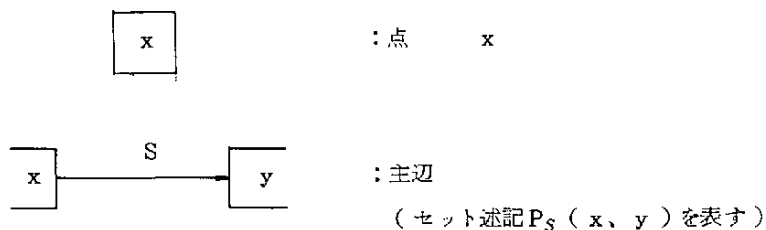
〔証明〕

CQGの定義より、任意のCODASYL問合せQを表すCODASYL問合せグラフ(CQG) Gが1つ存在する。逆に、任意のGに対して、Gの結合辺、主辺、制限辺の表す式の積を各々、結合式jf、S式sf、制限式rfとし、積 $\text{jf} \wedge \text{st} \wedge \text{rf}$ を問合せの条件式qualと出来る。又、Gの目標辺と目標式点($a_i = \text{exp}_i$)とが表す目標式 $a_i = \text{exp}_i$ の和を目標リストと出来る。よって、Gに対して、一意なCODASYL問合せが存在する。■

以降、論理CODASYLモデルにおける問合せとして、CQGを考えることにする。CQGに対して、以下の制約を設けることにする。

〔CQGの制約〕

全てのCODASYL問合せグラフ(CQG)内の各0点は主辺について連結である。□



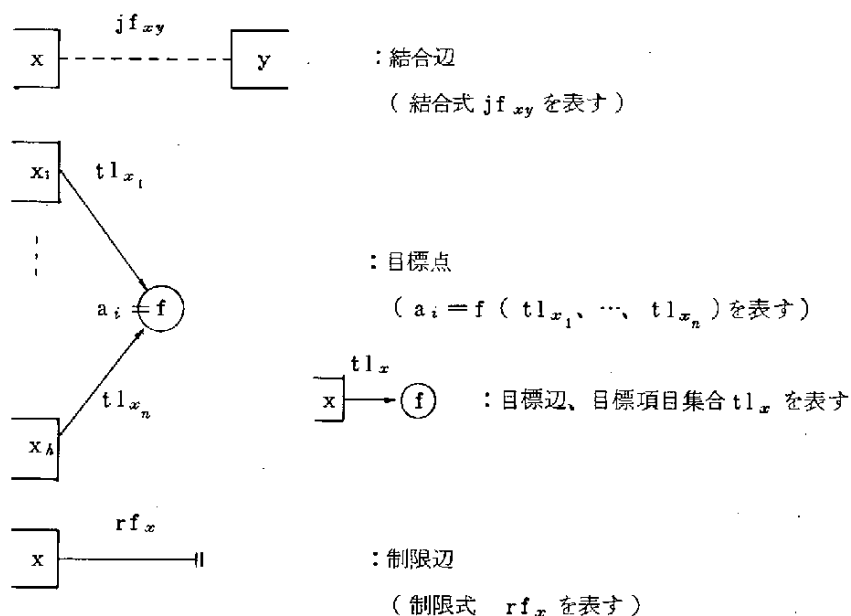


図 3.4 CQG 記法

CQG を表す為に、図 3.4 の様な記法を用いることにする。例として、A の CODASYL 問合せに対する CQG を図 3.5 に示す。CQG は、述語論理形式の非手続的な CQL 問合せの意味の直観的な理解を利用者に与えられる。この為、従来の CODASYL DBS 上の一般利用者インタフェースとして優れていると考えている。

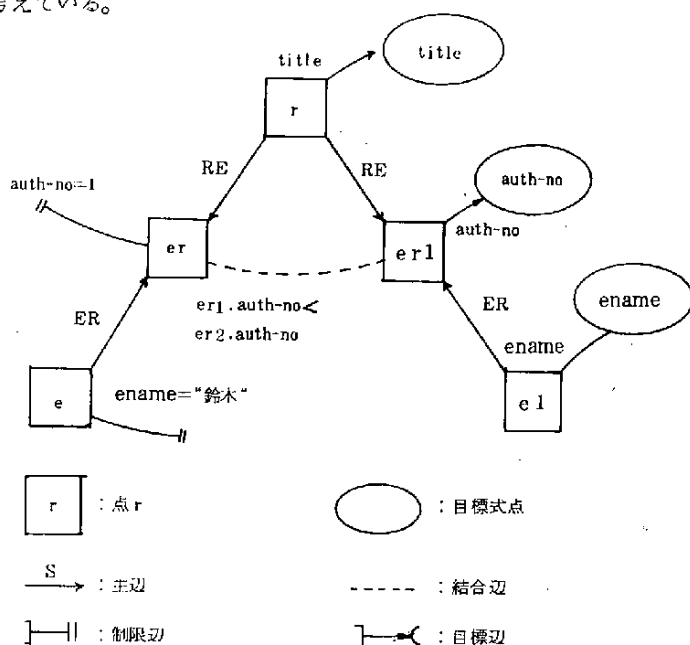


図 3.5 CODASYL 問合せグラフ (CQG)

3.2.3 論理操作演算—更新演算

論理データベースC上の更新演算について考える。まず、論理スキーマC内の各セット型は、型づけがなされているとする。この時、任意のレコード実現値(RO)集合をRとする。R内の各レコード実現値 $r \in R$ に対して、 r 及び、 r のM/A型のセット型 $S = (\underline{R}, \underline{R}', \underline{I}_S)$ の全ての子 $r' \in R'$ 、更に、 r' のM/A型セット型 S' の子 $r'' \in R'', \dots$ の集合を $mr(r)$ とする。各 $r' \in mr(r)$ を含む全てのセット実現値の集合を $ss(r')$ とし、 $\bigcup_{r' \in mr(r)} ss(r') = sr(r)$ とする。この時、

$$ms(r) \subseteq mr(r) \cup sr(r).$$

又、論理スキーマC内の各レコード型Rに対して、Rを子とする全てM/A型セット型 $S = (\underline{R}, \underline{R}', \underline{I}_S)$ の $(\underline{S}, \underline{R}')$ の対の集合を $O_S(R)$ とする。

A 基本更新演算

$\underline{R}, \underline{S} = (\underline{R}_1, \underline{R}_2, \underline{I}_S)$ を、論理スキーマC内の任意のレコード型とセット型とする。このとき、レコード実現値(RO)集合Rと、セット実現値(SO)集合Sとに対して、各々、実現値の消去、追加、置換の更新基本演算がある。以下に更新基本演算の意味を示す。

- (1) del (r, R); Rから r を除くとともに、 $ms(r)$ を論理データベースCから除く。
- (2) del (s, S); Rから r を除く。 $stype(\underline{S}) = M/A$ のとき、 $s = (\delta(r_1), \delta(r_2)) \in S$ とすると、 $ms(r_2)$ もCから除かれる。
- (3) app ($r, R \mid r_1 \in R_1, \dots, r_n \in R_n$); $O_S(\underline{R}) = \{(\underline{S}_1, \underline{R}_1), \dots, (\underline{S}_n, \underline{R}_n)\}$ とする。 r をRに加えるためには、まず、各 R_i 内にある r_i が存在せねばならない($i = 1, \dots, n$)。存在するならば、 r をRに加えると共に、各 S_i に $(\delta(r_i), \delta(r))$ を追加する。 r のdbキー $\delta(r)$ はDBSによって自動的にセットされるとする。
- (4) app ($s = (\delta(r_1), \delta(r_2)), S \mid r_1 \in R_1, r_2 \in R_2$); $\underline{S}$ はO/M型($stype(\underline{S}) = O/M$)でなければならない。追加しようとするセット実現値 $s = (\delta(r_1), \delta(r_2))$ は、 $r_1 \in R_1$ でかつ $r_2 \in R_2$ ならばSに追加される。
- (5) rep (r, r', R); R内のレコード実現値 r のデータ項目値を、 r' に置換する。
- (6) rep ($s = (\delta(r_1), \delta(r_2)), s' = (\delta(r'_1), \delta(r_2)), S \mid r'_1 \in R_1$); S内のセット実現値 $s = (\delta(r_1), \delta(r_2)) (\in S)$ の親を、ある $r'_1 \in R_1$ にかえる。

B CODASYL問合せ言語(CQL)

CQLにおける更新演算を定める。利用者は、CQLを用いて、更新対象を述語論理形式で非手続的に記述できる。ここで、RとSを各々、任意のRのレコード実現値(RO)集合、 $\underline{S} = (\underline{R}_1, \underline{R}_2,$

F_S) のセット実現値 (SO) 集合とする。

(1) R 内のレコード実現値の消去

```
var ( r, R ) ( r1, R1 ) ... ( rk, Rk ) ;  
delete r where qual ;
```

R_i は、C 内の任意の RO 集合である。この意味は、次の様である。

```
begin  
  T ← { r | r ∈ R ∧  $\bigwedge_{i=1}^h r_i ∈ R_i ∧ r ∈ R ∧ \text{qual}(r, r_1, \dots,$   
    rk ) } ;  
  
  for  $\forall r \in T$  do  
    del ( r, R )  
  
end ;
```

即ち、条件 qual を満足する全てのレコード実現値 $r \in R$ が、R から除かれる。同時に、 $ms(r)$ も論理データベース C から除かれる。図 3.3 のスキーマに対して、“Jipdec を消去せよ” は、次の様に書ける。

```
var ( o, ORG ) ;  
delete o where o. oname = “Jipdec” ;
```

“Jipdec” の消去は、その全ての職員 $e \in EMP$ が消され、更に、e とプロジェクトとの関連情報も PEL から除かれる。

(2) R への追加

```
var ( r1, R1 ) ... ( rn, Rn ) ( r1', R1' ) ... ( rk', Rk' ) ;  
append to R ( t1 ) where qual ;
```

ここで、 $O_S(R) = \{ (\underline{S}_1, \underline{R}_1), \dots, (\underline{S}_n, \underline{R}_n) \} (n \geq 0)$ とし、 R_i' を C 内の任意の RO 集合とする ($i = 1, \dots, h, h \geq 0$)。この意味は、次の様である。

[意味]

```
begin  
  T ← { ( v, r1, ..., rn ) | v = t1( r1, ..., rn, r1', ..., rk' )
```

$$\bigwedge_{i=1}^n r_i \in R_i \quad \bigwedge_{j=1}^h r_j \in R_j' \quad \wedge \quad \text{qual}(r_1, \dots, r_n, r_1', \dots, r_h')$$

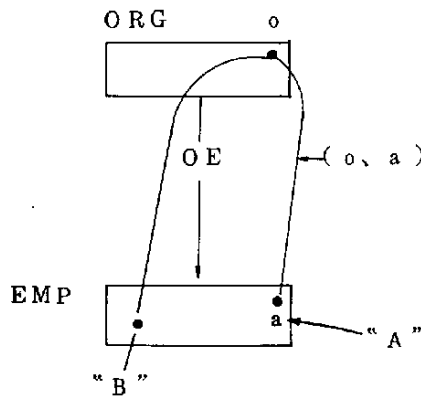
for $\forall (v, r_1, \dots, r_n) \in T$
do app (v, R | $r_1 \in R_1, \dots, r_n \in R_n$)
end ;

条件式 qual は、各 R_i に対してある r_i を定めねばならない。

[例]

例えば、図 3.3 のスキーマで、“A”を、新たにEMPに加える為には、qual 内で、Aの属することになる組織 (i.e. “B”と同じ組織) を明らかにせねばならない。

var (o, ORG) (e, EMP);
append to EMP (ename = “A”, ...) where
e. ename = “B” and OE(o, e);



(3) Rのレコード実現値のデータ項目値の置換

var (r, R) (r₁, R₁) ... (r_h, R_h);
replace r (t1) where qual ;

R_i は、C内の任意のRO集合である ($i=1, \dots, h$) この意味は、次の様である

[意味]

begin
 $T \leftarrow \{ (r, v) \mid v = t1(r, r_1, \dots, r_h) \bigwedge_{i=1}^h r_i \in R_i \wedge r \in R \wedge \text{qual}(r, r_1, \dots, r_h) \}$
end

```

for  $\forall (r, v) \in T$ 
    do    rep ( r, v, R )
end ;

```

[例]

例えば、Jipdecの職員の給料(sal)を10%上げる為には、次の様に書ける。

```

var ( o, ORG ) ( e, EMP ) ;
replace e ( sal = e. sal * 1.1 ) where
    o. oname = " Jipdec " and OE ( o, e ) ;

```

(4) Sのセット実現値の消去

```

var ( r2, R2 ) ( r1', R1' ) ... ( rh', Rh' ) ;
disconnect from S ( r2 ) where qual ;

```

R₂ は、Sの子RO集合である。R_i は、論理データベースC内の任意のRO集合である(i = 1, ..., h)。このCQLの意味は、次の様である。

[意味]

```

begin
    T ← { r2 | r2 ∈ R2  $\bigwedge_{i=1}^h$  ri' ∈ Ri'  $\wedge$  qual ( r2, r1', ..., rh' ) } ;
    for  $\forall r_2 \in T$ 
        do    del ( s, S | s = (  $\delta(r_1)$ ,  $\delta(r_2)$  ) )
    end ;

```

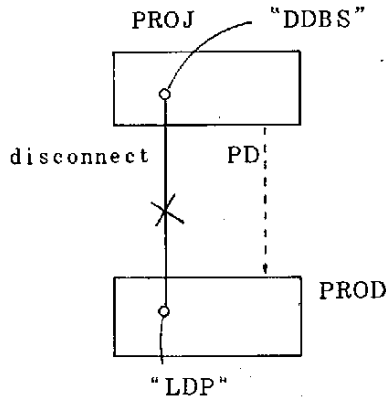
[例]

図3.3で、「ソフトウェアプロダクト「LDP」をDDBSプロジェクトから除く」は、S型PDはmc (PD) = O / Mなので次の様に書かれる。

```

var ( p, PROJ ) ( d, PROD ) ;
disconnect from PD ( d )
    where
        d. dname = " LDP " and
        p. pno = " DDBS " and
        PD ( p, d ) ;

```



(5) Sへの追加

```
var ( r1, R1 ) ( r2, R2 ) ( r1', R1' ) ... ( rk', Rk' );
connect to S ( r1, r2 ) where qual ;
```

ここで、SはO/M型でなければならない($\text{stype}(\underline{S}) = \text{O/M}$)。R₁とR₂は、各々Sの親と子レコード実現値(RO)集合である。R_i'はC内の任意のRO集合である。この意味は、次の様である。

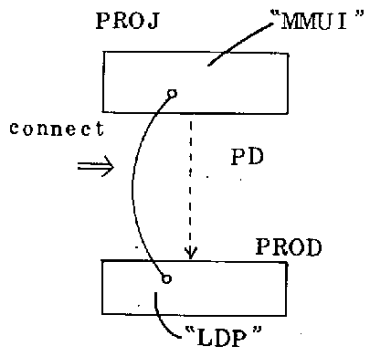
[意味]

```
begin
  T ← { ( r1, r2 ) | r1 ∈ R1 ∧ r2 ∈ R2 ∧  $\bigwedge_{i=1}^h$  ri' ∈ Ri' ∧ qual
    ( r1, r2, r1', ..., rk' ) } ;
  for ∀ ( r1, r2 ) ∈ T
    do app ( s = ( δ(r1), δ(r2) ), S | r1 ∈ R1, r2 ∈ R2 )
end ;
```

[例]

例えば、ソフトウェア製品“LDP”を、“MMUI”プロジェクトのものとするにはS型PDは $\text{mc}(\underline{PD}) = \text{O/M}$ なので次の様にかける。

```
var ( p, PROJ ) ( d, PROD );
connect to PD ( p, d ) where
  d. dname = "LDP" and p. pno = "MMUI" ;
```



(6) Sの置換

```
var (  $r_1, R_1$  ) (  $r_2, R_2$  ) (  $r_1', R_1'$  ) ... (  $r_k', R_k'$  ) ;
modify S (  $r_1, r_2$  ) where qual ;
```

R_1 と R_2 は各々、セット型 S の親と子 RO 集合である。 R_i' は、C 内の任意の RO 集合である。
この意味は次の様である。

[意味]

begin

$$T \leftarrow \{ (s' = (\delta(r_1'), \delta(r_2)), s = (\delta(r_1), \delta(r_2))) \mid$$

$$r_1' \in R_1 \wedge r_2 \in R_2 \wedge r_1 \in R_1 \bigwedge_{i=1}^k r_i' \in R_i' \wedge$$

$$\text{qual}(r_1, r_2, r_1', \dots, r_k') \} ;$$

for $\forall (s', s) \in T$ do

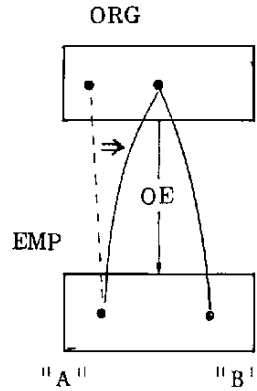
rep (s', s, S)

end ;

[例]

例えば、職員 "A" の所属を、"B" と同じにするには CQL で次の様に書ける。

```
var (  $e, EMP$  ) (  $o, ORG$  ) (  $e1, EMP$  ) ;
modify S (  $e, o$  ) where
   $e1.ename = "B"$  and  $e.ename = "A"$  and
   $OE(o, e1)$  ;
```



3.3 仮想 CODASYL DBS

本節では、論理層と物理層の間にある CODASYL DBS の仮想層について考える。仮想層の仮想データ構造を定義し、この上のデータ操作言語 (DML) [OLLET 78] の意味を、多テューリングマシンによって表す [TAKIM 83]。

3.3.1 仮想データ構造

3.2 で定義した論理データ構造 $C = (\underline{C}, C)$ の仮想層の表現を仮想データ構造 $V = (\underline{V}, V)$ とする。 $\underline{V}$ を論理スキーマ $\underline{C}$ に対する仮想スキーマ、 V を論理データベース C に対する仮想データベースとする。 V は、 C に対して、順序関係が導入されている。

まず、 $\underline{C}$ 内の各レコード型 $R = (\mathcal{Q}_R, \Gamma_R)$ に対して、 $\underline{V}$ 内の仮想レコード型 (又は、VR 型) $\underline{X} = (\mathcal{Q}_X, \Gamma_X, \langle X \rangle)$ が定義される。ここで、 $\underline{X}$ の項目集合 $\mathcal{Q}_X$ とインテグリティ条件 Γ_X とは、各々 $\mathcal{Q}_R$ と Γ_R と同じである ($\mathcal{Q}_X = \mathcal{Q}_R, \Gamma_X = \Gamma_R$)。レコード実現値 (RO) 集合 $R = \{ r \mid r \in \text{dom}(\mathcal{Q}_R) \wedge \Gamma_R(r) \}$ を、順序関係 $\langle X \rangle$ によって全順序づけたものを、仮想レコード実現値 (VRO) 集合 X とする。各レコード実現値 $r \in R$ と 1 対 1 対応する $x \in X$ を、 r の仮想レコード実現値 (又は、VR 実現値) とする。このとき、 $\forall r \in \mathcal{Q}_R (= \mathcal{Q}_X)$ に対して、 $\tau(r) = \tau(x)$ である。 X 内の VR 実現値は、その db キー δ によって全順序づけられる。即ち、

$$\forall x, x' \in X (x \neq x') \ x \langle_X x' \text{ iff } \delta(x) < \delta(x')$$

仮想データ構造 V では、 C とは独立に、db キーだけから成り、1 つの VR 実現値から成る VRO 集合 X_0 (スキーマ $\underline{X}_0 = (\mathcal{Q}_{X_0})$, $\mathcal{Q}_{X_0} = \{ \otimes X_0 \}$) を定義出来る。これを、システム¹⁾ VRO 集合と呼ぶ。 $\underline{X}_0$ をシステム VS 型とする。

論理スキーマ $\underline{C}$ 内の各セット型 $\underline{S} = (\underline{R}_1, \underline{R}_2, \Gamma_S)$ に対して、仮想スキーマ内に仮想セット型 (又は、VS 型) $\underline{Y} = (\underline{X}_1, \underline{X}_2, \Gamma_Y, \langle Y \rangle)$ を定義する。ここで、 $\underline{X}_1$ と $\underline{X}_2$ は、各々、レコード型 $\underline{R}_1$ と $\underline{R}_2$ に対応する仮想レコード型である。 Γ_Y は、 $\underline{Y}$ のインテグリティ条件で、 Γ_S と等しい

($\Gamma_Y = \Gamma_S$)。 $\underline{X}_1$ を $\underline{Y}$ の親、 $\underline{X}_2$ を $\underline{Y}$ の子という。ここで、各 $x_1 \in X_1$ に対して、以下を定義する。

$$m_Y(x_1) \triangleq \{x_2 \mid x_2 \in X_2 \wedge (x_1 \text{ と } x_2 \text{ は各々、 } r_1 \in R_1 \text{ と } r_2 \in R_2 \text{ の VR 実現値}) \wedge (r_1, r_2) \in S\}$$

$m_Y(x_1)$ を、 $x_1 \in X_1$ の子仮想レコード (VR) 実現値の集合とする。各 $x_2 \in m_Y(x_1)$ に対して $x_1 \in X_1$ を、 x_2 の親とする。この時、 $\underline{S}$ のセット実現値 (SO) 集合 S に対応した仮想セット実現値 (VSO) 集合 Y が次の様に定義される。

$$Y \triangleq \{(x_1, m_Y(x_1)) \mid x_1 \in X_1 \wedge m_Y(x_1) \neq \emptyset\}$$

各 $y = (x_1, m_Y(x_1)) \in Y$ を、仮想セット (VS) 実現値とする。集合 $m_Y(x_1)$ は、全順序づけられている。順序関係 $<_Y$ としては、次の3種がある。

- (i) あるデータ項目 $t \in \pi_{X_2}$ に対して、 $m_Y(x_1)$ 内の VR 実現値を昇順に順序づける。
- (ii) (i) に対して、降順に順序づける。
- (iii) DBS が自動的に順序づける。
- (i) 又は (ii) におけるデータ項目 t を、 VS 型 $\underline{Y}$ の整列データ項目とする。

任意の仮想レコード (VR) 型 $\underline{X} = (\Omega_X, \Gamma_X, \langle X \rangle)$ の仮想レコード実現値 (VRO) 集合 X が、あるデータ項目 $t \in \pi_X$ による全順序関係を持つとき、 $\underline{X}_0$ と $\underline{X}$ とに仮想セット (VS) 型 $\underline{Y} = (\underline{X}_0, \underline{X}, \Gamma_Y, \langle Y \rangle)$ を定義できる。この仮想セット実現値 (VSO) 集合 Y は、 $Y \triangleq \{(x_0, X)\}$ である。この様な VS 型は、単項 VS 型と呼ばれる。¹⁾ Y を単項 VSO 集合とする。

ここで、各 VS 型 $\underline{Y} = (\underline{X}_1, \underline{X}_2)$ に対して、次の述語 $P_Y: \text{dom}(\Omega_{X_1}) \times \text{dom}(\Omega_{X_2}) \rightarrow \{\text{true}, \text{false}\}$ を定義する。

$$P_Y(x_1, x_2) = \begin{cases} \text{true} & \text{if } x_1 \in X_1 \wedge x_2 \in X_2 \wedge x_2 \in m_Y(x_1) \\ \text{false} & \text{otherwise} \end{cases}$$

仮想レコード実現値 (VRO) 集合と、仮想セット実現値 (VSO) 集合との集合を、仮想スキーマ $\underline{V}$ の仮想データベース V とする。

以上より、論理データ構造 C と仮想データ構造 V との間には、以下の性質が存在することは明らかである。

〔性質3〕

仮想データ構造 V から、順序関係、システム VR 型と VRO 集合、及び単項 VS 型と VSO 集合を除いたものは、論理データ構造 C である。□

3.3.2 仮想CODASYLマシン (VCM)

仮想データベース V の操作を行うデータ操作言語 (DML) [OLLET 78] について考える。DML の意味を明らかにする為に、DML を直接的に実行出来る仮想的な CODASYL マシン (VCM) を、多テープチューリングマシンで表す。VCM は、有限制御 F と無限本のテープ集合 T とから成る [図 3.6]

A テープ

各テープ T ($\in T$) は、セルで区切られた無限長テープである。又、 T は、一時に 1 つのセルを見れるヘッド H_T を持っている。 T は、データベース (DB) テープ、プログラムテープ、出力テープの 3 種のテープから成る。仮想スキーマ V は、 n (≥ 1) 個の仮想レコード (VR) 型 $X_1, \dots, X_n$ と、 m (≥ 0) 個の仮想セット (VS) 型 $Y_1, \dots, Y_m$ から成るとする。DB テープは、各 VR 型 X の仮想レコード実現値 (VRO) 集合 X を格納する為の n (≥ 1) 本の仮想レコード (VR) テー

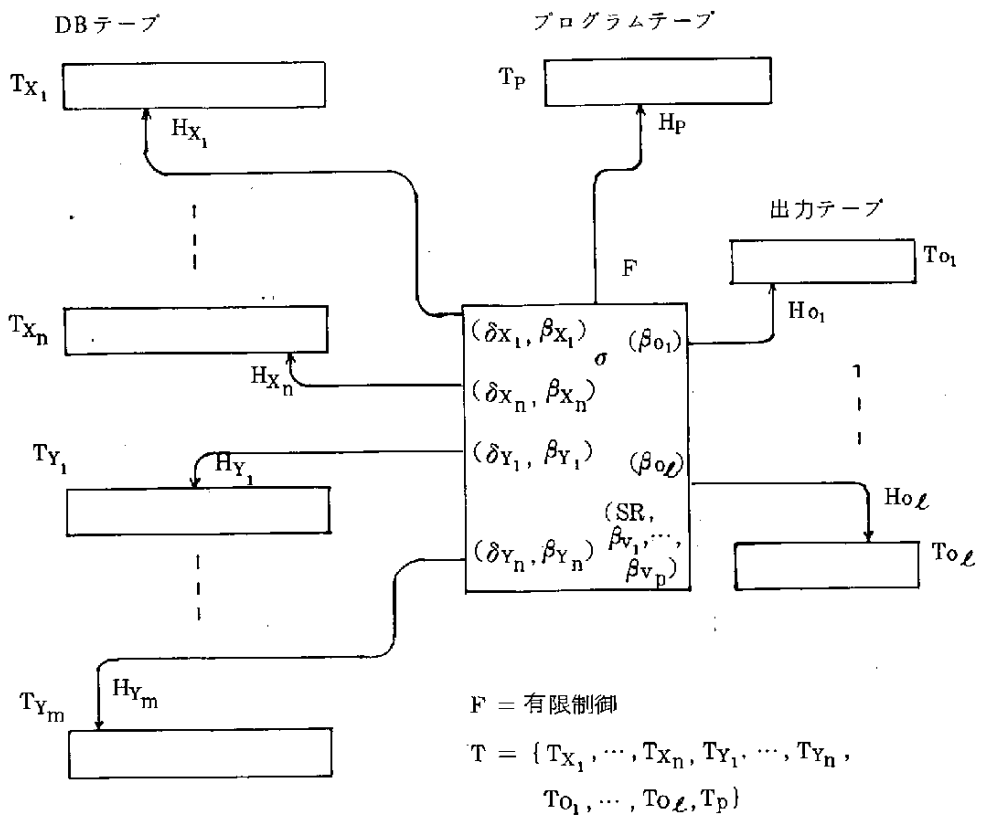


図 3.6 VCM の構成

ブ T_X と、各VS型 $\underline{Y}$ の仮想セット実現値(VSO)集合 Y を格納する為の $m(\geq 0)$ 本の仮想セット(VS)テーブル T_Y とから成る。DBテーブルの連続した各セルには、左から右に、VR型又は、VS型の順序関係に基づいて、VR実現値が書かれる。連続したVR実現値の書かれたセルの左右端には、各々記号 $\vdash$ と $\dashv$ が書かれ、他の全てのセルには空白記号 δ が書かれている。VRO集合 $X = \{x_1, \dots, x_q\}$ (ここで、 $x_1 <_X x_2 <_X \dots <_X x_q$) を考える。このテーブル表示は、図3.7のようになる。これを $ID_{T_X} = \vdash x_1 x_2 \dots x_q \dashv$ と表す。テーブル T_X のテーブル記号の集合を T_X とすると、

$$T_X \quad \begin{array}{|c|c|c|c|c|c|c|c|c|c|} \hline \delta & \delta & \vdash & x_1 & x_2 & \dots & x_q & \dashv & \delta & \delta \\ \hline \end{array}$$

図3.7 Xのテーブル表示

$T_X \triangleq \text{dom}(\mathcal{Q}_X) \cup \{\vdash, \dashv, \delta\}$ である。

次に、VS型 $\underline{Y} = (X_1, X_2, \Gamma_Y, \langle \gamma \rangle)$ のVSO集合を $Y = \{(x_{1i}, x_{2i2}, \dots, x_{2iq_i}) \mid x_{1i} \in X_1\}$ (ここで $x_{2i1} <_\gamma x_{2i2} <_\gamma \dots <_\gamma x_{2iq_i}$ ($q_i \geq 1$)) とする。この時、テーブル T_Y 上では、図3.8の様に表される。各仮想セット(VS)実現値間には、区切り記号 $;$ が書かれる。テーブル T_Y のテーブル記号の集合を T_Y とすると、 $T_Y \triangleq \text{dom}(\mathcal{Q}_{X_1}) \times \text{dom}(\mathcal{Q}_{X_2}) \cup \{\vdash, \dashv, ;, \delta\}$ である。

$$T_Y \quad \begin{array}{|c|} \hline \delta & \vdash & x_{11} & x_{211} & \dots & x_{21q_1} & ; & x_{12} & x_{221} & \dots & x_{22q_2} & ; & \dots & ; & x_{1k} & x_{2k1} & \dots & x_{2kq_k} & \dashv & \delta \\ \hline \end{array}$$

図3.8 Yのテーブル表示

プログラムテーブル T_P は、読取り専用テーブルである。 T_P の連続した各セルには、左から右にVCM演算記号が書かれる。VCM演算記号を、表3.2にまとめてある。 T_P を T_P のテーブル記号集合とする。

出力テーブル T_{0_i} ($i=1, \dots, l, l \geq 1$) には、検索の目標集合、中間目標集合の組が、各セルに書かれる。 T_{0_i} をテーブル T_{0_i} のテーブル記号集合とする。

B VCMの動作

仮想CODASYLマシン(VCM)の有限制御 F は、次の有限記憶と次状態関数 σ から成っている。

- (1) 各DBテーブル T 毎に、記憶領域 β_T と δ_T とを持つ。 δ_T には、常にテーブル T のヘッド H_T 下のセルのVR実現値のdbキーが、 H_T の移動のたびにセットされる。 β_T は、 T の1セル分の記憶領域である。 δ_T を、テーブル T の現在子、 β_T を T の作業域と言う。 δ_T と β_T とを各々、 δ_T と β_T の定義域とする。
- (2) 各出力テーブル T 。毎に、1セル分の作業域 β_0 を持つ。 β_0 を、 β_0 の定義域とする。

(3) VCM演算の実行結果状態がセットされる状態レジスタSRをFは持つ。SRをSRの定義域とすると、 $SR = \{ S(\text{uccess}), F(\text{ailure}) \}$ 。

(4) 他に、Fは有限個の記憶領域 β_{v_i} ($i=1, \dots, p$) を持つ。 β_{v_i} を β_{v_i} の定義域とする。

VCMは、各テープTのヘッド H_T 下のテープ記号と、有限制御Fの内部状態によって、内部状態を変えるときともに、 H_T を動作(m_T)させる。 H_T の可能な動作の集合を M_T とすると、 $M_T = \{ L, R, N, LT, RT, DA, RP, AP, DL, RD, AC \}$ である。

(i) ヘッド H_T を1つ左のセルに動かす(L)。

(ii) ヘッド H_T を1つ右のセルに動かす(R)。

(iii) ヘッド H_T を動かさない。

(iv) ヘッド H_T を、テープTの最左のVR実現値の書かれたセルに移動する(LT)。

(v) H_T を最右のVR実現値の書かれたセルに移動させる(RT)。

(vi) δ_T 内のdbキー値を持つセル上に H_T を動かす(DA)。このようなセルがT内に存在しない時は、 $\neg$ 上にセルを動かし、 $\delta_T = \emptyset$ とする。

(vii) H_T 下のセルを、 β_T の値によって書き換える(RP)。Tが、DBテープの時には、セル内のデータ項目値のみを書き換えられる。

(viii) H_T 下に、新たに β_T の値が書かれたセルを追加する(AP)。TがVRテープの場合には、追加されるセルのdbキーは、Fが自動的に決めるものとする。

(ix) H_T 下のセルを除く(DL)。

(x) H_T 下のセルを、 β_T に書き出す(RD)。

(xi) Tが、VRO集合Xを格納する為のVRテープで、あるデータ項目 $t \in \pi_X$ について連想機能を有している時、 H_T を β_T 内のt値を持つセルの中の一番左のものへ動かす(AC)。ない時には、 H_T を $\neg$ に動かし、 δ_T には、 $\emptyset$ がセットされる。tを、CALC項目¹⁾とする。

VCMは、7字組(Q, T, B, D, σ , q_0 , F)で定式化出来る。Qは、有限制御の内部状態の集合である。 q_0 は、初期状態である($q_0 \in Q$)。Fは、終了状態の集合である($F \subseteq Q$)。

T, B, Dは次の様である。

$$T \triangleq \bigcup_{i=1}^n T_{X_i} \quad \bigcup_{j=1}^m T_{Y_j} \times T_P \quad \bigcup_{t=1}^e T_{O_t}$$

$$B \triangleq \bigcup_{i=1}^n \beta_{X_i} \quad \bigcup_{j=1}^m \beta_{Y_j} \quad \bigcup_{t=1}^1 \beta_{O_t} \times SR \quad \bigcup_{u=1}^p \beta_{v_u}$$

$$D \triangleq \bigcup_{i=1}^n \Delta_{X_i} \quad \bigcup_{j=1}^m \Delta_{Y_j}$$

T は、各テープ T_{X_i} 、 T_{Y_j} 、 T_P 、 T_{O_i} のテープ記号集合の直積である。 D はDBテープの現在子の定義域の直積である。

σ は次状態関数で、 $\sigma: Q \times T \times B \times D \rightarrow Q \times T \times D \times M$ である。ここで

$$M = \prod_{i=1}^n M_{X_i} \times \prod_{j=1}^m M_{Y_j} \times M_P \times \prod_{t=1}^p M_{O_t}$$

M_{X_i} 、 M_{Y_j} 、 M_P 、 M_{O_t} は、各々テープ T_{X_i} 、 T_{Y_j} 、 T_P 、 T_{O_t} のヘッド H_{X_i} 、 H_{Y_j} 、 H_P 、 H_{O_t} の可能な動作集合 ($\subseteq \{L, R, N, H, T, DA, RP, DL, RD, AC\}$) である。各時刻でのVCMの様相 (configuration) は、 (q, t, b, d, i) (ここで、 $q \in Q$ 、 $t \in T$ 、 $b \in B$ 、 $d \in D$) として表せる。ここで、 i は各テープ T のヘッド H_T の位置の組である。状態 $q \in Q$ でテープ T のあるセル t_i にヘッド H_T があるとき、これをテープ T の時刻表示 $ID_T \triangleq \vdash t_1 \cdots t_{i-1} q t_i \cdots t_k \dashv$ と表す。ある時刻の様相 (q, t, b, d, i) が、次状態関数 $\sigma(q, t, b, d) = (q', t', b', d', m)$ によって、様相 (q', t', b', d', i') に変化したとする。このとき各テープ T の表示は、 T の H_T の動作 m_T ($\in m$) によって ID_T から $ID_{T'}$ に変化する。これを、 $ID_T \vdash \text{---} ID_{T'}$ と表す。表3.1に動作の様相の変化をまとめる。

表3.1 テープ(T)のヘッド(H_T)の動作

$(ID_T, \delta_T, \beta_T) \vdash \text{---} (ID_{T'}, \delta_{T'}, \beta_{T'})$

m_T	δ_T	β_T	$\delta_{T'}$	$\beta_{T'}$	$ID_{T'}$	説明
N	$\delta(t_i)$	/	$\delta(t_i)$	/	$\vdash t_1 \cdots q' t_i \cdots t_k \dashv$	H_T を動かさない
R	$\delta(t_i)$	/	$\delta(t_{i+1})$	/	$\vdash t_1 \cdots t_i q' t_{i+1} \cdots t_k \dashv$	H_T を1つ右のセルに動かす
L	$\delta(t_i)$	/	$\delta(t_{i-1})$	/	$\vdash t_1 \cdots q' t_{i-1} t_i \cdots t_k \dashv$	H_T を1つ左のセルに動かす
H	$\delta(t_i)$	/	$\delta(t_1)$	/	$\vdash q' t_1 \cdots t_k \dashv$	H_T を最左のセルに動かす
T	$\delta(t_i)$	/	$\delta(t_k)$	/	$\vdash t_1 \cdots q' t_k \dashv$	H_T を最右のセルに動かす
DA	k	/	$\delta(t_j) = k$	/	$\vdash t_1 \cdots q' t_j \cdots t_k \dashv$	H_T を $\delta(t_i) = k$ なる一番左のセル t_i に動かす
	k	/	$\emptyset$	/	$\vdash t_1 \cdots t_k q' \dashv$	ないときに、 H_T を $\dashv$ に動かす
RP	$\delta(t_i)$	v	$\delta(v)$	/	$\vdash t_1 \cdots t_{i-1} q' v \cdots t_k \dashv$	H_T 下のセル値をvとする
AP	$\delta(t_i)$	v	$\delta(v)$	/	$\vdash t_1 \cdots t_{i-1} q' v t_i \cdots t_k \dashv$	H_T 下に、セルvを追加する
DL	$\delta(t_i)$	/	$\delta(t_{i+1})$	/	$\vdash t_1 \cdots t_{i-1} q' t_{i+1} \cdots t_k \dashv$	H_T 下のセルを除く
RD	$\delta(t_i)$	/	$\delta(t_i)$	t_i	$\vdash t_1 \cdots q' t_i \cdots t_k \dashv$	H_T 下のセルを β_T に書く
AC		v	$\delta(t_j)$	/	$\vdash t_1 \cdots q' t_j \cdots t_k \dashv$	$t(t_j) = t(v)$ なる一番左のセルに H_T を動かす
		v	$\emptyset$	/	$\vdash t_1 \cdots t_k q' \dashv$	ないとき、 H_T を $\dashv$ に動かす

VCMは、プログラムテープ T_p のヘッド H_p 下のセル内のVCM演算記号を読んで、各テープ T_{X_i} ($i=1, \dots, n$), T_{Y_j} ($j=1, \dots, m$), T_{O_t} ($t=1, \dots, l$)に必要な動作をさせる。各VCM演算の意味は、テープヘッドの動作によって表せる。表3.2には、各VCM演算の意味をまとめる。又、表3.2は、同時に、COBOL DML[OLLET78]が、VCM演算に1対1対応することを示している。この表は、DMLの意味を表しており、CODASYL DBSの実現の基礎となるものである。以上より次の命題が成り立つことは明らかである。

[命題3.2]

全てのCOBOL DMLを、仮想CODASYLマシン(VCM)上でシミュレートできる。■

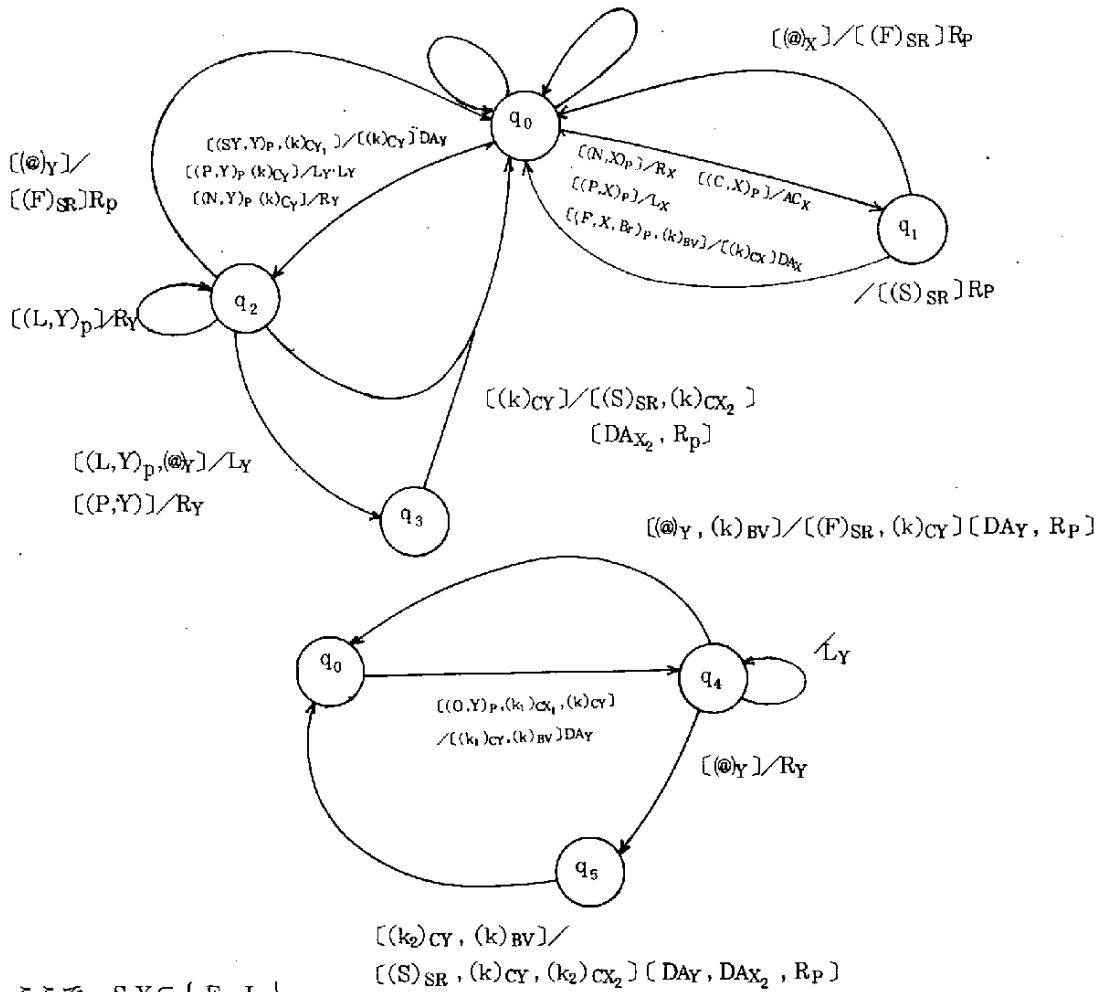
表3.2 VCM演算とCOBOL DML

VCM演算	意 味	COBOL DML
<u>first</u> (X);	$m_x = H$	<u>find first</u> X <u>within</u> A. ^{*)}
<u>last</u> (X);	$m_x = T$	<u>find last</u> X <u>within</u> A. ^{*)}
<u>next</u> (X);	$m_x = R$	<u>find next</u> X <u>within</u> A. ^{*)}
<u>prior</u> (X);	$m_x = L$	<u>find prior</u> X <u>within</u> A. ^{*)}
<u>CALC</u> (X);	$m_x = AC$	<u>find any</u> X.
<u>first</u> (Y);	H_{X_1} 下のセルを親とする T_Y 内の先頭の子に H_Y を動かし、 H_{X_2} もこのセルに動かす	<u>find first</u> X ₂ <u>within</u> Y.
<u>last</u> (Y);	H_{X_1} 下のセルを親とする T_Y 内の最後の子に H_Y を動かし、 H_{X_2} もこのセルに動かす	<u>find last</u> X ₂ <u>within</u> Y.
<u>next</u> (Y);	H_Y を1つ右へ(R)動かし、 H_{X_2} もこのセルに動かす	<u>find next</u> X ₂ <u>within</u> Y.
<u>prior</u> (Y);	H_Y を1つ左へ(L)動かし、 H_{X_2} もこのセルに動かす	<u>find prior</u> X ₂ <u>within</u> Y.
<u>owner</u> (Y);	H_Y の親のセルに H_{X_1} を動かす	<u>find owner</u> <u>within</u> Y.
<u>get</u> (X);	$m_x = RD$	<u>get</u> X.
<u>if</u> SR=n, <u>go to</u> L;	SR=nならば、ラベルLの演算に H_p を動かす	<u>if</u> SR=n <u>go to</u> L.
<u>open</u> (T _i);	$m_{oi} = T$	<u>open</u> T.
<u>close</u> (T _i);	$m_{oi} = T$	<u>close</u> T.
<u>write</u> (T _i);	$m_{oi} = AP$	<u>write</u> T.
<u>stop</u> ;	VCMの停止	<u>stop</u> .
<u>accept</u> (X, βv);	$\beta v \leftarrow \delta_X$;	<u>accept</u> βv <u>from</u> X <u>current</u> .
<u>accept</u> (Y, βv);	$\beta v \leftarrow \delta_Y$;	<u>accept</u> βv <u>from</u> Y <u>current</u> .
<u>find</u> (X, βv);	H_X を、 βv 内のdbキーを持つセルに動かす	<u>find</u> X; <u>db-key</u> <u>is</u> βv .

VCM演算	意 味	COBOL DML
<u>store</u> (X);	β_x の値を、 $\{x$ を満たす様に T_x に加える	<u>store</u> X.
<u>erase</u> (X);	H_x 下のセル(x)を含む全てのセルをVSテープ T_y から消す。xが親の時は、 T_y からxを親とする全てのセルを消す。 H_x 下のセルを消す	<u>erase</u> X.
<u>modify</u> (X);	$m_x = RP$	<u>modify</u> X <u>including</u> $Y_1, \dots, Y_n$.
<u>connect</u> (Y);	H_{x_1} と H_{x_2} 下のセルを、親子対として T_y に $\{y$ を満たす様に加える	<u>connect</u> X_2 <u>to</u> Y.
<u>disconnect</u> (Y);	H_{x_2} 下のセルを T_y から除く	<u>disconnect</u> X_2 <u>from</u> Y.
<u>modify</u> (Y);	H_{x_2} 下のセルの親を、 H_{x_1} 下のセルとする。	<u>modify</u> X_2 <u>only</u> Y.
$P(\beta_1, \dots, \beta_n)$	内の記憶領域 $\beta_1, \dots, \beta_n$ 内の値で比較式 f を評価し、真又は偽とする	$P(\beta_1, \dots, \beta_n)$
<u>go to</u> L ;	ラベルLのセルに H_p を動かす	<u>go to</u> L.
$\beta_i \leftarrow \beta_j ;$	β_i の内容を β_j に移す	<u>move</u> β_j <u>to</u> β_i

$[(R, T)_P, (@)_O] / [(F)_{SR}] R_P$
 $[(W, T)_P, (@)_{BO}] / [(F)_{SR}] R_P$
 $[(O, Y)_P, (@)_{X_2}] / [(F)_{SR}] R_P$
 $[(C, X)_P, (@)_{BX}] / [(F)_{SR}] R_P$
 $[(SY, Y)_P, (@)_{X_1}] / [(F)_{SR}] R_P$
 $[(SY, Y)_P, (@)_Y] / [(F)_{SR}] R_P$
 $[(CX, X)_P, (@)_X] / [(F)_{SR}] R_P$

$[(R, T)_P, (v)_O] / [(v)_{BO}] RD_O$
 $[(W, T)_P, (v)_{BO}] / [(S)_{SR}] AP_O$
 $[(CL, T)_P] / TH_O$
 $[(OP, T)_P] / TH_O$
 $[(A, X, BV) (k)_{CX}] / [(k)_{BR}] R_P$
 $[(G, X)_P, (x)_X] / [(S)_{SR}, (x)_{BX}] R_P$
 $[(F, X)_P] / [(S)_{SR}] [R_P, TH_X]$
 $[(L, X)_P] / [(S)_{SR}] [R_P, TT_X]$



ここで $SY \in \{F, L\}$

$SY \in \{N, P\}$

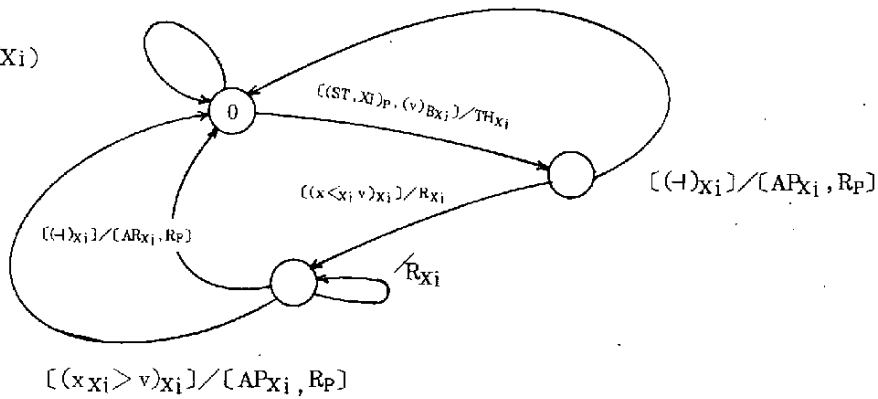
$CX \in \{F, L, N, P, G\}$

$@ \in \{+, -, :\}$

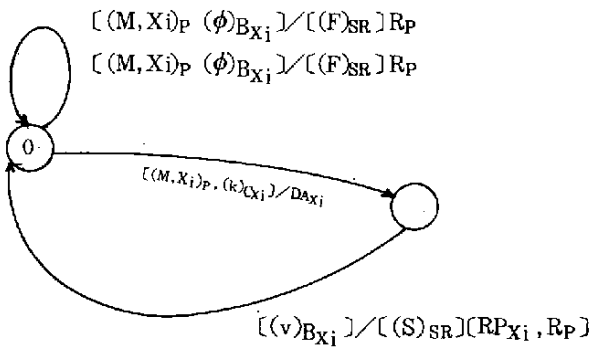
図 3.9 VCM の状態遷移図

$$[(ST, Xi)_P, (\phi)_{BX_i}] / [(F)_{SR}] R_P$$

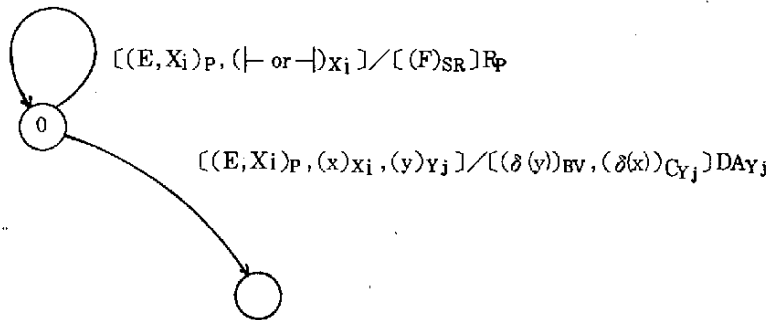
store (Xi)



modify (Xi)



erase (Xi)



3.4 概念 CODASYL モデル

CODASYL DBSの最上位層である概念層について論じる。概念層では、論理データ構造を制約した概念データ構造と、概念データ構造上の基本操作演算について論じる。

3.4.1 概念データ構造

論理データ構造 C は、レコード型と、レコード型間の部分関数を表すセット型とから成る論理スキーマ $\underline{C}$ と、レコード型とセット型の各々に、実際に値を与えたレコード実現値 (RO) 集合とセット実現値 (SO) 集合とから成る論理データベース C との対である ($C = (\underline{C}, C)$)。論理スキーマ $\underline{C}$ 内のある n (≥ 2) 個のレコード型 $\underline{R}_1, \dots, \underline{R}_n$ (互いに異っている必要はない) が、共通の子レコード型 $\underline{R}$ を持つ n 個のセット型 $\underline{S}_i = (\underline{R}_i, \underline{R}, \Gamma_{S_i})$ の親であり、かつ部分関数 $f: \text{dom}(\mathcal{Q}_{R_1}) \times \dots \times \text{dom}(\mathcal{Q}_{R_n}) \rightarrow \text{dom}(\mathcal{Q}_R)$ が存在しているとする。このとき、概念データ構造では、論理データ構造の2つのレコード型間の部分関数に加えて、この n 個のレコード型間の関係も表される。概念層では、このセット型 $\underline{S}_i$ ($i = 1, \dots, n$) と $\underline{R}$ とをまとめてリンク型 $\underline{L} = (\underline{R}_1, \underline{S}_1(\underline{R}_1, \underline{R}), \dots, \underline{R}_n, \underline{S}_n(\underline{R}_n, \underline{R}), \underline{R}, \Gamma_L)$ とする [図 3.10]。 Γ_L は、 $\mathcal{Q}_{R_1}, \dots, \mathcal{Q}_{R_n}, \mathcal{Q}_R$ から意味的に導出されるインテグリティ条件である。リンク型 $\underline{L}$ に、実際に値を与えたものを、リンク実現値 (LO) 集合 L とする。

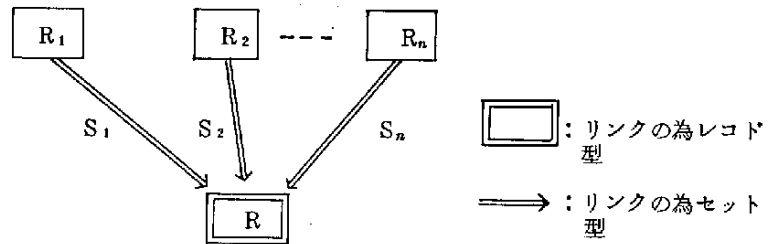


図 3.10 リンク型 $\underline{L} = \{ \underline{R}, \underline{S}_1(\underline{R}_1, \underline{R}), \dots, \underline{S}_n(\underline{R}_n, \underline{R}) \}$

$$L \triangleq \{ l \mid l = (r_1, \dots, r_n, r) \in R_1 \times \dots \times R_n \times R \mid \bigwedge_{i=1}^n (r_i, r) \in S_i \wedge \Gamma_L(r_1, \dots, r_n, r) \}$$

各 $l \in L$ をリンク (L) 実現値とする。 $\underline{L}$ を表すセット型 $\underline{S}_i$ をリンクの為のセット型、各 $\underline{S}_i$ の共通の子レコード型 $\underline{R}$ をリンクの為のレコード型とする。よって、概念スキーマ $\underline{D}$ は、レコード型 $\underline{R}$ 、セット型 $\underline{S}$ 、及びリンク型 $\underline{L}$ の3種の型から成る。概念データベース D も $\underline{R}$ のレコード実現値 (RO) 集合、 $\underline{S}$ のセット実現値 (SO) 集合、 $\underline{L}$ のリンク実現値 (LO) 集合の3種の実現値集合から成る。

ここで、全てのリンクの為のセット型の型は、 $M/A(\text{stype}(\underline{S}) = M/A)$ であるとする。^{*}

^{*} [CODAS73] では R 型 $\underline{R}_1$ と $\underline{R}_2$ 間に複数の S 型 $\underline{S}_i(\underline{R}_1, \underline{R}_2)$ ($i = 1, \dots, n, n \geq 2$) あるとき、 $mc(\underline{S}_i) = M/A$ であるのは高々1つとなる。そうでないと、各 S 型 $\underline{S}_i$ は皆同一の親子関係をもつことになってしまう。

〔命題3.3〕

全ての概念CODASYLデータ構造は、論理CODASYLデータ構造で表せる。

〔証明〕

概念CODASYLデータ構造のリンク型 $\underline{L}$ は、セット型とレコード型で表せるので明らかである。■

図3.3において、レコード型 $\underline{PEL}$ とセット型 $\underline{PE}$ と $\underline{EP}$ とは、部分関数 $f_{PEL} : \text{dom}(\mathcal{Q}_{EMP}) \times \text{dom}(\mathcal{Q}_{PROJ}) \rightarrow \text{dom}(\mathcal{Q}_{PEL})$ を表すので、リンク型 $\underline{PEL} = \{ \underline{EMP}, \underline{PE} = (\underline{EMP}, \underline{PEL}), \underline{PROJ}, \underline{PE} = (\underline{PROJ}, \underline{PEL}), \underline{PEL} \}$ を表す。同様に、レコード型 $\underline{REL}$ とセット型 $\underline{RE}$ と $\underline{ER}$ も、リンク型を表している。リンクの為のレコード型2重箱で又リンクの為のセット型を2重矢で表すと、図3.11の様になる。

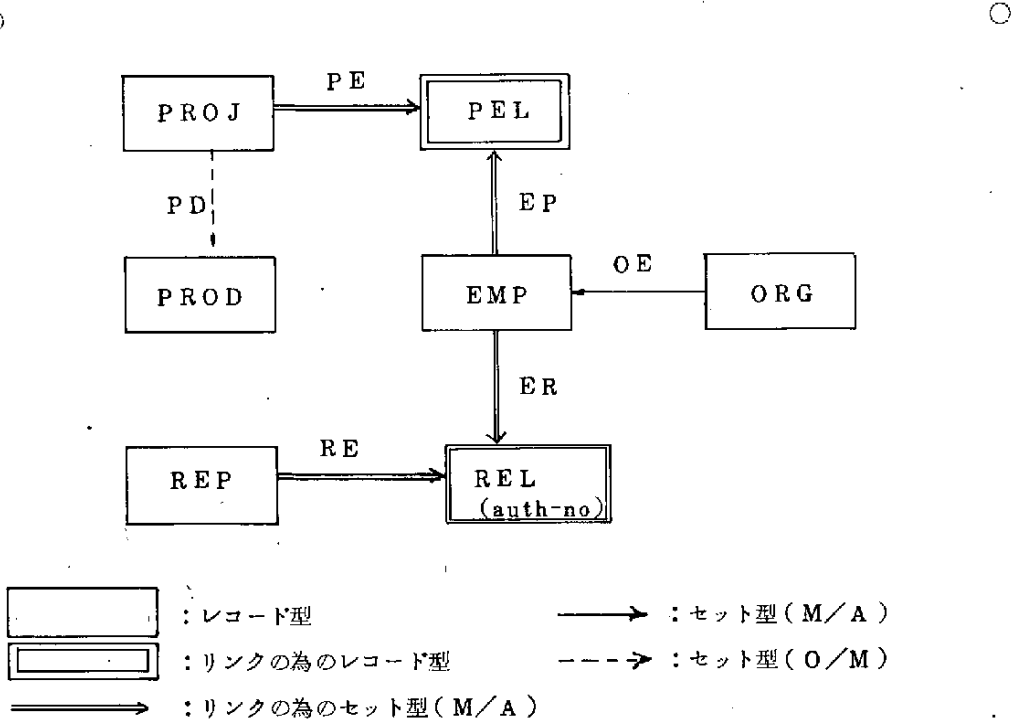


図3.11 概念CODASYLスキーマ

3.4.2 基本操作演算

データベースD内の任意のRO集合をRとする。各 $r \in R$ に対して、その全てのM/A型S実現値の子 $r' \in R'$ の集合が定まる。同様に $r' \in R'$ のM/A型S実現値の子 $r'' \in R''$ の集合が定まる。 r から始めて、この様に再帰的に定義されていく全てのR実現値の集合を $M(r, R)$ とする。又 $M(r, R)$ 内の各 $r' \in R'$ を親又は子とする全てのS実現値の集合を $O(r')$ とする。この時、 $ms(r, R) \triangleq M(r, R) \cup O(r')$ 。又、D内の各レコード型 $\underline{R}$ に対して、 $\underline{R}$ を子とする全ての $\forall r' \in M(r, R)$

のM/Aセット型 $\underline{S'}$ とその親 $\underline{R'}$ の対の集合を $O_S(R)$ とする。 $R, S(R_1, R_2), \{L, S_i(R_i, L) (i=1, \dots, n)\}$ を各々、D内の任意のRO、SO、LO集合とすると、次の9種の論理演算がC上に存在する。

- (1) $\underline{del}(r, R \mid r \in R)$; $ms(r, R)$ をDから消去する。
- (2) $\underline{del}(s, S \mid s \in S)$; $s = (k_1, k_2) \in S$ の時、 $mc(\underline{S}) = M/A$ ならば、 $ms(r_2, R_2) (dbk(r_2) = k_2)$ を、 O/M ならば s のみを消去する。
- (3) $\underline{del}(l, L \mid l \in L)$; $l \in L$ 及び、各 S_i から l を子とする S 実現値 S_i を消去する。
- (4) $\underline{app}(r, R \mid r_1 \in R_1, \dots, r_n \in R_n)$; 全ての $(S_i, R_i) \in O_S(R)$ に対して、ある $r_i \in R_i$ の時、 r を R に加えると共に、各 S_i に S 実現値 $(\delta(r_1), \delta(r))$ を加える。
- (5) $\underline{app}(s, S \mid r_1 \in R_1, r_2 \in R_2)$; $mc(\underline{S}) = O/M$ の時、 $r_1 \in R_1, r_2 \in R_2$ ならば、 S に $(\delta(r_1), \delta(r_2))$ を追加する。
- (6) $\underline{app}(l, L \mid r_1 \in R_1, \dots, r_n \in R_n)$; R を L とした(4)と同じ。
- (7) $\underline{rep}(r, r', R \mid r \in R)$; $r \in R$ のデータ項目値を r' に置換する。
- (8) $\underline{rep}(s, s', S \mid s \in S)$; $s = (k_1, k_2) \in S$ を $s' = (k_1', k_2) (k_1' \neq k_1)$ に置換する。即ち、子に対する親を置換する。
- (9) $\underline{rep}(l, l', L \mid l \in L)$; $l \in L$ のデータ項目値を l' に置換する。

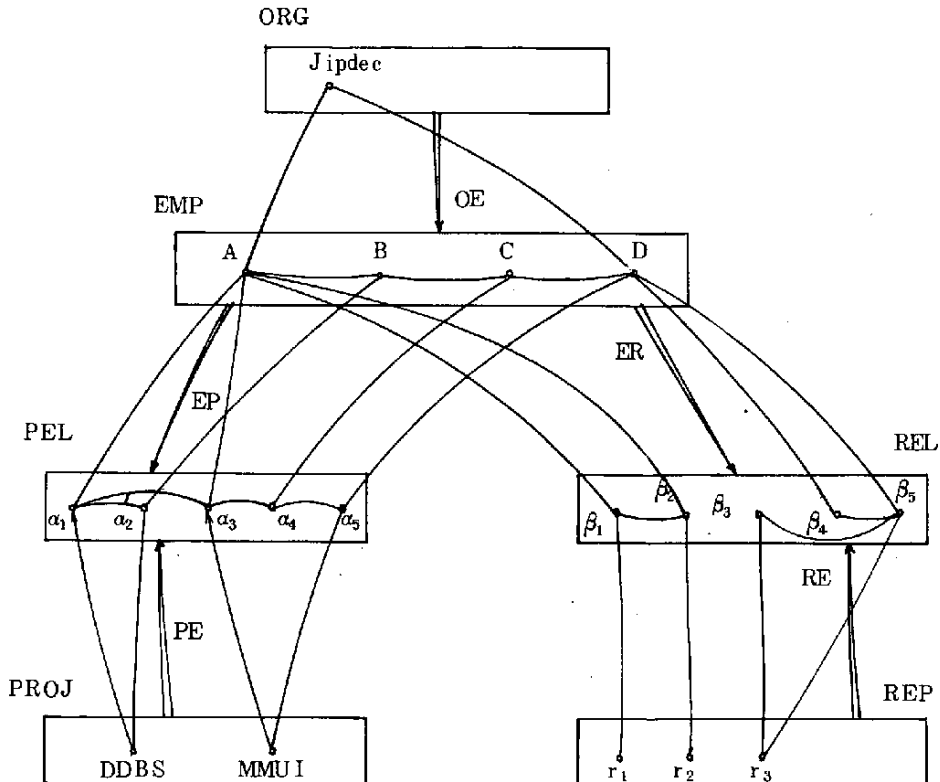


図 3.1 2 実現値の関連

図3.11について考える。図3.12は、図3.11のVR実現値、VS実現値を示している。ここで、 $\text{stype}(\underline{OE}) = \text{stype}(\underline{EP}) = \text{stype}(\underline{PE}) = \text{stype}(\underline{ER}) = \text{stype}(\underline{RE}) = M/A$ なので、

$$\begin{aligned} \text{ms}("A", \text{EMP}) = \{ & "A", ("A", \alpha_1), ("A", \alpha_3), \alpha_1, \alpha_3, ("A", \beta_1), ("A", \beta_2), \\ & (\text{DDBS}, \alpha_1), (\text{MMUI}, \alpha_3), \\ & \beta_1, \beta_2, (r_1, \beta_1), (r_2, \beta_2) \} \end{aligned}$$

よって、 $"A" \in \text{EMP}$ を消去すると、データベースからVR実現値 $\alpha_1, \alpha_3 \in \text{PEL}$ 、 $\beta_1, \beta_2 \in \text{REL}$ 及びVS実現値 $("A", \alpha_3) \in \text{EP}$ 、 $(\text{DDBS}, \alpha_1), (\text{MMUI}, \alpha_3) \in \text{PE}$ 、 $(r_1, \beta_1), (r_2, \beta_2) \in \text{RE}$ 、 $("A", \beta_1), ("A", \beta_2) \in \text{ER}$ も同時に消去される。同時に、 $\text{Jipdec} \in \text{ORG}$ に対して

$$\begin{aligned} \text{ms}(\text{Jipdec}, \text{ORG}) = \{ & \text{Jipdec}, (\text{Jipdec}, "A"), (\text{Jipdec}, "B"), \\ & (\text{Jipdec}, "C"), (\text{Jipdec}, "D") \} \\ \cup & \text{ms}("A", \text{EMP}) \quad \cup \text{ms}("B", \text{EMP}) \\ \cup & \text{ms}("C", \text{EMP}) \quad \cup \text{ms}("D", \text{EMP}) \end{aligned}$$

である。

3.5 物理CODASYLモデル

仮想の下層物理層について考える。

3.5.1 物理データ構造

物理データ構造は、仮想データ構造 $V = (\underline{V}, V)$ に対して、物理格納空間PS上のVR実現値の格納場所及びVRO集合、VSO集合内のVR実現値間の順序関係の実現方法が定義される。

物理格納空間PSは、連続した格納単位の集合で、各格納単位は一連の番地が与えられている。PSは、連続した格納単位の部分集合(エリア)によって分割されている。エリアは更に、エリア毎に固定数の連続した格納単位から成る論理ページに分割されている。

仮想データ構造 $V = (\underline{V}, V)$ に対する物理データ構造 $P = (\underline{P}, P)$ は、VのPS上の表現である。各VR型 $\underline{X} = (\mathcal{Q}_X, \Gamma_X, \{x\})$ に対して、物理レコード(PR)型 $\underline{T} = (\mathcal{Q}_T, \Gamma_T, A, L)$ が定義される。ここで $\mathcal{Q}_T = \mathcal{Q}_X$ 、 $\Gamma_T = \Gamma_X$ である。Aは、PS内のエリア名、Lは格納方法で、 $L \in \{N, C_\tau, U\}$ である($\tau \in \{\mathcal{Q}_T - \otimes T\}$)、Uはある物理セット(PS)型)。VRO集合Xに対して、物理レコード実現値集合(PRO)集合Tが定まる。X内のVR実現値と、T内の物理レコード(PR)実現値とが1対1対応する。各 $t \in T$ はPS上のエリアA内のある論理ページ内の格納単位に連続して格納される。この時、 $\otimes T$ の値は、論理ページ番号と、論理ページの先頭の格納単

位から、 t を格納した格納単位までの相対単位数 (番地) となる。 $k = @T(t)$ とすると、 $page(k)$ は t を格納したページ番号を表し、 $loc(k)$ はページ内の相対番地を表すとする。ここで、VR型のdb キー $@X$ と、 $@T$ を対応させる。よって、VR型の順序関係 $\langle_X$ は、PRO集合 T が定まった時に定まる。各 $t \in T$ の $@T$ (番地) は、 L によって定まる。

- (1) $L=N$ の時、自動的に決める。
- (2) $L=C_T$ の時、ある項目 $\tau \in \{Q_T - @T\}$ に対して、各 $t \in T$ の $@T(t)$ は $\tau(t)$ のあるハッシュ関数 $hash$ 値となる ($@T(t) = hash(\tau(t))$)。
- (3) $L=U$ の時は、PS型で述べる。

次に、VS型 $\underline{Y} = (\underline{X}_1, \underline{X}_2, \Gamma_Y, \langle_Y)$ に対して、物理セット (PS) 型 $\underline{U} = (\underline{T}_1, \underline{T}_2, \Gamma_U, \langle_U, PT)$ が定まる。ここで $\Gamma_U = \Gamma_Y$ 、 $\langle_U = \langle_Y$ であり、 $\underline{T}_1, \underline{T}_2$ は各々、VR型 $\underline{X}_1, \underline{X}_2$ のPR型である。この時、PR型 $\underline{X}_2$ の $L_2 = U$ であるとする。

[クラスタリング]

- (1) $A_1 = A_2$ の時、全ての $t_1 \in T_1$ に対して、全ての t_1 の子 $t_2 \in T_2$ は、同一論理ページ内になるべく格納される。クラスタリングされる。

$$\begin{aligned} \forall t_1 \in T_1 \quad \forall t_2 \in T_2 (t_2 \text{ は } t_1 \text{ の子}) \\ page(k_1) = page(k_2) \quad \wedge \\ k_1 = @T_1(t_1) \quad \wedge \quad k_2 = @T_2(t_2) \end{aligned}$$

- (2) $A_1 \neq A_2$ の時、全ての $t_1 \in T_1$ に対する全ての子 $t_2 \in T_2$ は、同一の論理ページ内に格納される。

$$\begin{aligned} \forall t_1 \in T_1 \quad \forall t_2, t_2' \in T_2 (t_2 \text{ は } t_1 \text{ の子}) (t_2 \neq t_2') \\ page(k_2) = page(k_2') \quad \wedge \\ k_2 = @T_2(t_2) \quad \wedge \quad k_2' = @T_2(t_2') \quad \square \end{aligned}$$

PS型 $\underline{U}$ の順序関係 $\langle_U$ は、P実現値のポインタで実現される。ここで、全順序集合 $A = \{a_1, \dots, a_n\}$ ($n \geq 0$ 、 $a_1 \langle_U a_2 \langle_U \dots \langle_U a_n$) に対して、次の3種のポインタを定める。

[ポインタ]

- (1) $a_1, a_2 \in A$ ($a_1 \neq a_2$) に対して、 $f_A(a_1) = a_2$
- (2) $\forall a_1, a_n \in A$ ($n > 1$) $l_A(a_1) = a_n$
- (3) $\forall a_i, a_{i+1} \in A$ ($2 \leq i \leq n-1$) に対して、

$$n_A(a_i) = a_{i+1}$$

$$p_A(a_{i+1}) = a_i$$

$$(4) \quad \forall a_i \in A \quad (i \geq 1)$$

$$O_A(a_i) = a_1 \quad \square$$

この時、VS実現値に対応してU内の各物理セット実現値($t_1, m_U(t_1)$)に対して、次のポイントが定めれる($m_U(t_1)$ は、 $m_Y(x_1)$ に対応した t_1 の子PR実現値の集合)。

$PT \in \{N, NP, NO, NOP\}$ である。

- (1) $PT=N$ の時 f_u, n_u だけ
- (2) $PT=NP$ f_u, n_u, l_u, p_u だけ
- (3) $PT=NO$ f_u, n_u, O_u だけ
- (4) $PT=NOP$ f_u, n_u, l_u, p_u, O_u

これらのポイントは、各PR実現値 $t_1 \in T_1, t_2 \in T_2$ が有まる。

3.5.2 物理操作演算

物理データ構造上の演算は、物理空間の論理ページを読み、その中を順にサーチ。又はポイントを順々にたどる操作となる。格納はある論理ページ内の格納単位への値のセットと、ポイントのセットとなる。消去は、ポイントの消去となる。

4. 概念モデルによるCODASYLモデル表現

概念モデルのR関係を型づけ、型毎に更新演算の意味(範囲、条件)を与える事によって、CODASYLモデルと等価なモデル(ローカル概念(LC)モデル)が構成出来る事を以下に示す。

4.1 LCモデルのデータ構造

[定義]

概念データ構造のR関係スキームを以下の様に特性化したものをローカル概念(LC)データ構造と呼ぶ。

- (1) R関係スキームRとして、BとG型の2種を設ける。RがE関係E₂からE₁への部分関数を表す時、B型とする。 $\otimes E_2$ の値によってR内の組を一意に識別出来るので、 $\otimes E_2$ は主キーとなる。これを $R(\otimes E_1, \otimes E_2)$ と記す。 $R = (\underline{E}_1, \dots, \underline{E}_n, \underline{Q}_R', \Gamma_R)$ が部分関数、 $\bigcup_{i=1}^n \text{dom}(\underline{Q}_{E_i}) \rightarrow \text{dom}(\underline{Q}_R')$ を表す時、RをG型とす。この時、R内の各組は主属性集合 $\{\otimes E_1, \dots, \otimes E_n\}$ 値によって識別できるので、主属性集合が主キーとなる。これを $R(\otimes E_1, \dots, \otimes E_n, a, \dots, a_m)$ と記す。
- (2) 更に、B型のRには、T(全)とP(部分)の2つ型があり、更新に対して異なった制約を与える〔4.2を参照〕。

以上の特性を持つスキーマをローカル概念スキーマ(LCS)と、これに実際の値を与えたものをローカル概念(LC)データベースとする。□

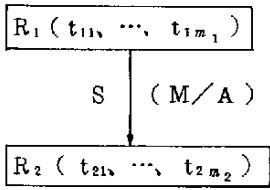
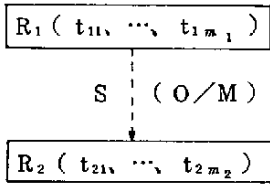
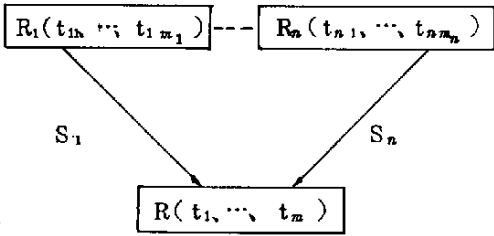
LCモデルと概念CODASYLモデルのデータ構造は、次の様に対応づけられる〔表4.1〕

[LCデータ構造と概念CODASYLデータ構造の対応規則]

- (1) レコード型 $R(t_1, \dots, t_m)$ とE関係スキーム $R(\otimes R, t_1, \dots, t_m)$ が、データ項目 t_i と属性 t_i 、dbキーと主属性 $\otimes R$ の様に対応する。任意のE関係スキーム $R(\otimes R, \dots)$ と $R'(\otimes R', \dots)$ ($R \neq R'$)に対して、 $\text{dom}(\otimes R) \neq \text{dom}(\otimes R')$ である。
- (2) セット型 $S(\underline{R}_1, \underline{R}_2)$ とB型R関係スキーム $S(\otimes R_1, \otimes R_2)$ が対応する。ここで、 $\otimes R_i$ はレコード型 R_i に対応したE関係スキームの主キーである($i=1, 2$)。SO集合Sは R_2 から R_1 への部分関数であるので、この関係をそのままR関係に対応づける。この時部分関数の値域を表す $\otimes R_2$ が主キーとなる。スキームは、セット型がM/Aの時T型に、O/Mの時P型になる。
- (3) リンク型 $L = \{R(t_1, \dots, t_m), \underline{R}_i, S_i(\underline{R}_i, \underline{R}) \mid (i=1, \dots, n) \mid (n \geq 2, m \geq 0)\}$ とG型R関係スキーム $L(\otimes R_1, \dots, \otimes R_n, t_1, \dots, t_m)$ が対応する。^{*} $\otimes R_i$ はレコード型 R_i に対応したE関係スキームの主キーである($i=1, \dots, n$)。LO集合 $L = \{L, S_1, \dots,$

^{*} $\otimes R_i = \otimes R_j$ ($i \neq j$) の時には、 $S_i \otimes R_i, S_j \otimes R_j$ を属性名とする。

表 4.1 LC と概念 CODASYL データ構造の対応

概念 CODASYL データ構造	LC データ構造
レコード $\underline{R}(t_1, \dots, t_m)$	E 関係スキーム $\underline{R}(\otimes \underline{R}, t_1, \dots, t_m)$
M/A セット型 	E 関係スキーム $\underline{R}_1(\otimes \underline{R}_1, t_{11}, \dots, t_{1m_1})$ $\underline{R}_2(\otimes \underline{R}_2, t_{21}, \dots, t_{2m_2})$ T 関係スキーム $\underline{S}(\otimes \underline{R}_1, \otimes \underline{R}_2)$
O/M セット型 	E 関係スキーム $\underline{R}_1(\otimes \underline{R}_1, t_{11}, \dots, t_{1m_1})$ $\underline{R}_2(\otimes \underline{R}_2, t_{21}, \dots, t_{2m_2})$ P 関係スキーム $\underline{S}(\otimes \underline{R}_1, \otimes \underline{R}_2)$
リンク型 	E 関係スキーム $\underline{R}_i(\otimes \underline{R}_i, t_{i1}, \dots, t_{im_i})$ $(i = 1, \dots, n)$ G 関係スキーム $\underline{L}([\underline{S}_1] \otimes \underline{R}_1, \dots, [\underline{S}_n] \otimes \underline{R}_n, t_1, \dots, t_m)$

S_n は、 R から $R_1 \times \dots \times R_n$ への単射であり、主属性集合 $\{\otimes \underline{R}_1, \dots, \otimes \underline{R}_n\}$ によって $L \times R \times \dots \times R_n$ の各組を識別出来るので、これが主キーとなる。□

以上の対応規則より、次の命題が成り立つ事は明らかである。

〔命題 4.1〕

任意の概念 CODASYL データ構造と、(1)~(3)によって導かれる LC データ構造とは 1 対 1 対応し、データ構造は等価である。■

図4.1は、図3.1.1のローカル概念(LC)スキーマを示している。例えば、リング型 $\underline{PEL} = \{ \underline{ER} = (\underline{EMP}, \underline{REL}), \underline{RE} = (\underline{REP}, \underline{REL}), \underline{REL} \}$ に対して、G関係スキーマ $\underline{PEL} (\textcircled{E}, \textcircled{R}, \text{auth-no})$ が対応する。

E関係スキーマ $\underline{EMP} (\textcircled{E}, \text{eno}, \text{ename}, \dots)$
 $\underline{PROJ} (\textcircled{P}, \text{pno}, \text{pname}, \dots)$
 $\underline{REP} (\textcircled{R}, \text{rno}, \text{title}, \dots, \text{cost}, \dots)$
 $\underline{ORG} (\textcircled{O}, \text{oname}, \dots)$
 T関係スキーマ $\underline{OE} (\textcircled{O}, \textcircled{E})$
 P関係スキーマ $\underline{PD} (\textcircled{P}, \textcircled{D})$
 G関係スキーマ $\underline{PEL} (\textcircled{P}, \textcircled{E}, \text{position})$
 $\underline{REL} (\textcircled{R}, \textcircled{E}, \text{auth-no})$

図4.1 図3.1.1のLCスキーマ

4.2 LCモデルの操作演算

本節では、ローカル概念(LC)モデルの操作演算について考える。まず、LCデータ構造上の基本更新演算を定義する。次に、LC操作言語(LCML)としての関係問合せ言語(RQL)の構文を定義し、その意味を、基本更新演算によって表す。

4.2.1 LCモデルの基本演算

LCデータベース $\mathcal{L}$ 内の任意のE関係を R (スキーマを $\underline{R} = (\mathcal{R}_R = \{ \textcircled{R}, t_1, \dots \})$)とし、E関係 R 内の各組を r とする。命題4.1より、LCデータ構造とCODASYLデータ構造は1対1対応するので、集合 $t_S(r, R)$ と $T_S(\underline{R})$ を、各々3.2の $ms(r, R)$ 、 $O_S(\underline{R})$ と同一に定義出来る。 $R(\textcircled{R}, t_1, \dots)$ 、 $S(\textcircled{R}_1, \textcircled{R}_2)$ 、 $L(\textcircled{R}_1, \dots, \textcircled{R}_n, b_1, \dots)$ を各々、 $\mathcal{L}$ 内の任意のF、B、G型関係とする。データ構造が特性化されたので、CML[2.3]に対して次の9種の基本演算が存在する。

[LC基本更新演算]

- (1) $\underline{del}(r, R \mid r \in R)$; から $t_S(r, R)$ を消去する。
- (2) $\underline{del}(s, S \mid s \in S)$; S がT型の時、 $\textcircled{R}_2(s) = \textcircled{R}_2(r_2)$ なる $t_S(r_2, R_2)$ を消去する。P型の時、 s のみを消去する。
- (3) $\underline{del}(l, L \mid l \in L)$; 組 $l \in L$ を消去する。
- (4) $\underline{app}(r, R \mid r_1 \in R_1, \dots, r_n \in R_n)$; 全ての $(\underline{S}_i, \underline{R}_i) \in T_S(\underline{R})$ に対して、ある組 $r_i \in R_i$ の時、 R に r を加えると共に $\textcircled{R}_i(r_i) = \textcircled{R}_i(s_i) \wedge \textcircled{R}(s_i) = \textcircled{R}(r)$ なる s_i を S_i に加える($i = 1, \dots, n$)。

- (5) $\text{app}(s, S \mid r_1 \in R_1, r_2 \in R_2)$; S が P 型で、 $r_1 \in R_1, r_2 \in R_2$ の時 $\oplus R_1(r_1) = \oplus R_1(s) \wedge \oplus R_2(s) = \oplus R_2(r_2)$ なる組 s を S に加える。
- (6) $\text{app}(l, L \mid r_1 \in R_1, \dots, r_n \in R_n)$; 全ての R_i に対して、ある $r_i \in R_i$ の時、 $\oplus R_i(r_i) = \oplus R_i(l)$ ($i = 1, \dots, n$) なる組 l を L に加える。
- (7) $\text{rep}(r, r', R \mid r \in R)$; (8) $\text{rep}(s, s', S \mid s \in S)$; (9) $\text{rep}(l, l', L \mid l \in L)$; 各々、 $r \in R, s \in S, l \in L$ の非主キー値を r', s', l' で置換する。 $\square$

[命題 4.2]

LC の(1)~(9)と、CODASYL の(1)~(9)の基本演算は、この順で1対1対応する。

[証明]

命題1より、集合 $ms(r, R)$ と $ts(r, R)$ 、 $O_S(\underline{R})$ と $T_S(\underline{R})$ とは互いに1対1対応する。よって、LC の(i)とCODASYL の(i)の基本演算とは、等価な更新条件と更新効果を持つ事 ($i = 1, \dots, 9$) は、明らかである。■

命題4.1と命題4.2より、次の定理が成り立つ。

[定理 4.1]

LCモデルと概念CODASYLモデルとは1対1対応し、等価である。■

4.2.2 LC操作言語(LCML)—RQL検索

A RQL

LCデータベース上の操作言語(LCML)について考える。LCMLとして、述語論理に基づき、関係計算言語QUEL[HELDG75]と類似した関係問合せ言語RQL(relational query language)を、本節で示す。まず検索について述べる。RQLによるLCデータベースに対する問合せ(LC問合せとする)は、次の様に書かれる。

$$\begin{aligned} & \text{range } (x_1, X_1) \dots (x_l, X_l); \\ & \text{retrieve into } T (a_1 = \text{exp}_1, \dots, a_k = \text{exp}_k) \dots \dots \dots (1) \\ & \text{where } \quad \quad \text{qual}; \end{aligned}$$

$X_1, \dots, X_l$ ($l \geq 1$) は、LCスキーマ内の任意の1個の関係スキーム名である(但し、互いに異っている必要はない)。 x_i は、関係スキーム X_i に対する組変数であり、 x_i の定義域は関係 X_i となる ($i = 1, \dots, l$)。

LC問合せ(1)式内の関係 R の属性 $a \in \mathcal{Q}_R$ は、 r, a (r は R の組変数)と表される。 T は、LC問合せの目標関係スキーム $\underline{T} = (\mathcal{Q}_T = \{a_1, \dots, a_k\})$ の名前である。($a_1 = \text{exp}_1, \dots, a_k = \text{exp}_k$) は、目標リスト($t1$)と呼ばれる。各 $a_i = \text{exp}_i$ を目標式とする。各属性 $a_i \in \mathcal{Q}_T$ を、目標属性とする。 exp_i は、定数、属性、又はこれらの上の算術式(e.g. $x, a + g.b/5$)

である。各式 exp_i に対して、 $\text{var}(\text{exp}_i)$ は exp_i の参照する組変数の集合であり、 $\text{tatt}(\text{exp}_i)$ は、 exp_i 内の属性の集合を表すとする。ここで、 $\text{var}(\text{exp}_i) = \{x_{i1}, \dots, x_{i\ell_i}\} \subseteq \{x_1, \dots, x_\ell\}$ とすると、 exp_i は、関数 $\text{exp}_i : \text{dom}(\mathcal{Q}_{x_{i1}}) \times \dots \times \text{dom}(\mathcal{Q}_{x_{i\ell_i}})$ である。又、目標リスト $t1$ は、関数 $t1 : \text{dom}(\mathcal{Q}_{x_1}) \times \dots \times \text{dom}(\mathcal{Q}_{x_\ell}) \rightarrow \text{dom}(\mathcal{Q}_T)$ である。各組変数 x に対して、 $t1_x \triangleq \{x.a \mid x.a \in \bigcup_{i=1}^1 \text{tatt}(\text{exp}_i)\}$ を x の出力属性集合とする。

qual は、条件式 (qualification) である。 qual は、次の様に、積正規化されている。

```

qual ::= rf and jf and pf
rf    ::= rfxi1 and ..... and rfxiτi
rfx  ::= rpx | rpx and rfx | rpx or rfx
rpx  ::= expx θ v
expx ::= x.a | <x上の算術式>
θ      ::= <| ≤ | = | ≥ | > | ≠

```

rf は、問合せの制限式である。 rf_x は、組変数 x の制限式であり、 $\text{rf}_x : \text{dom}(\mathcal{Q}_X) \rightarrow \{\text{true}, \text{false}\}$ である。 exp_x は、 x についての算術式である。

```

jf    ::= jfxi1xj1 and ..... and jfxiqxjq
jfxy ::= jpxy | jpxy and jfxy | jpxy or jfxy
jpxy ::= expx θ expy

```

x と y は組変数である。 jf は、問合せの結合式、 jf_{xy} は、組変数 x と y 間の結合式であり、 $\text{jf}_{xy} : \text{dom}(\mathcal{Q}_X) \times \text{dom}(\mathcal{Q}_Y) \rightarrow \{\text{true}, \text{false}\}$ である。

```

pf    ::= ppxi1xj1 and ..... and ppxiqxjq
ppxy ::= x.⊗R θ' y.⊗R
θ'    ::= = | ≠

```

pf は、問合せの主結合式である。 pp_{xy} は、組変数 x と y 間の主結合式である。 $\otimes R$ は、 x と y が表す関係の主属性である。主属性値は、識別子であるので、比較演算子としては、 $=$ と $\neq$ のみが許される。又、 x と y の表す関係の少くとも一方は R 関係でなければならない。〔4.1 節の対応規則(1)を参照されたい〕。 pp_{xy} は、述語 $\text{pp}_{xy} : \text{dom}(\otimes R) \times \text{dom}(\otimes R) \rightarrow \{\text{true}, \text{false}\}$ である。以上より、条件式 qual は、 $\text{qual} : \text{dom}(\mathcal{Q}_{x_1}) \times \dots \times \text{dom}(\mathcal{Q}_{x_\ell}) \rightarrow \{\text{true}, \text{false}\}$ である。

LC問合せ(1)の意味は、次の様である。

$$T = \{ t \mid t = t1(t_1, \dots, t_l) \wedge \\ (t_1, \dots, t_l) \in X_1 \times \dots \times X_l \wedge \\ qual(t_1, \dots, t_l) \}$$

[例]

例として、図4.1のLCスキーマに対する問合せ「鈴木を第1著者とする論文の中で、共著者がいるものの番号(rno)と、共著者及び共著番号(auth-no)を求めよ」は、RQLで次の様書ける。

```
range(e, EMP)(e1, EMP)(r, REP)(er, ERL)(er1, ERL);
retrieve into T(r.rno, er1.auth-no, e1.ename)
where
e.ename = "鈴木" and e.@E=er.@E and
er.auth-no = 1 and er.@R=r.@R and
r.@R=er1.@R and er1.@E=e1.@E and
er.auth-no < er1.auth-no ; ..... (2) □
```

B LC問合せグラフ(LQG)

(1)の形式のLCS問合せをQとする。Qのグラフの表現としてのLC問合せグラフ(LQG)は次の様に定義される。

[定義]

Q[(1)]に対するLCS問合せグラフ(LQG) $G=(N, E)$ は、点集合Nと辺集合Eとから成る。点には、組(変数)点と、目標式点との2種がある。LCS問合せQの各組変数 x_i に対して、組(変数)点 x_i が対応する。Qの目標リスト(t1)内の目標式 $a_i = exp_i$ に対して、目標式点に対応し、式 $a_i = exp_i$ が付される。

辺には、2点に接する辺と、1点のみに接する辺とがある。前者には、無向の主辺と結合辺、有向の目標辺があり、後者には、制限辺がある。問合せ内の結合式 jf の積の要素 jf_{xy} に対して、組点xとy間に存在し、結合式 jf_{xy} が付加される。主結合式 pf 内の積の要素 pp_{xy} に対して、組点xとy間の主辺が対応し、主結合述語 pp_{xy} が付される。目標リスト(t1)内の各目標式 $a_i = exp_i$ に対して、各組点 $x \in var(exp_i)$ から目標式点 $a_i = exp_i$ に有向な目標辺が対応し、xの出力属性集合 tl_x が付される。問合せの制限式 rf の積の要素の制限式 rf_x に対して、組点xに接する制限辺が対応し、制限式 rf_x が付される。□

LQGは、図4.2の記法を用いて表される。

〔定義〕

目標辺に接する組点を目標点とする。結合辺に接する組点を、結合点とする。LCS問合せグラフ(LQG)内の目標点と結合点の集合を出力点集合Oとし、O内の各組点を出力点とする。□

□ : (E)組点(E関係スキーム)

◻ : (R)組点(R(T, P, G)関係スキーム)

$\boxed{x} \xrightarrow{\oplus R} [\quad]$: 主辺($x, \oplus R = y, \oplus R$)

$\boxed{x} \xrightarrow{\oplus R} \diagup [y]$: 主辺($x, \oplus R \neq y, \oplus R$)

$\boxed{x} \xrightarrow{jf_{xy}} [y]$: 結合辺

$\boxed{x} \xrightarrow{rf_x} ||$: 制限辺

$\bigcirc \begin{matrix} a_i = \\ \text{exp}_i \end{matrix}$: 目標式点

$\boxed{x} \xrightarrow{tl_x} \bigcirc a_i = \text{exp}_i$: 目標辺

図4.2 CQG記法

〔命題4.3〕

RQLによる全てのLC問合せと、LCS問合せグラフ(LQG)とは1対1対応する。

〔証明〕

LQGの定義より明らかである。■

本論文では、LQGに対して、以下の制約を設ける。

〔制約〕

LQG内の全ての組点は、主辺について連結でなければならない。又、主辺としては、主属性間の等(=)結合を表すもののみを考える。□

〔例〕

例として、AのLCS問合せ(2)に対するLQGを図4.3に示す。□

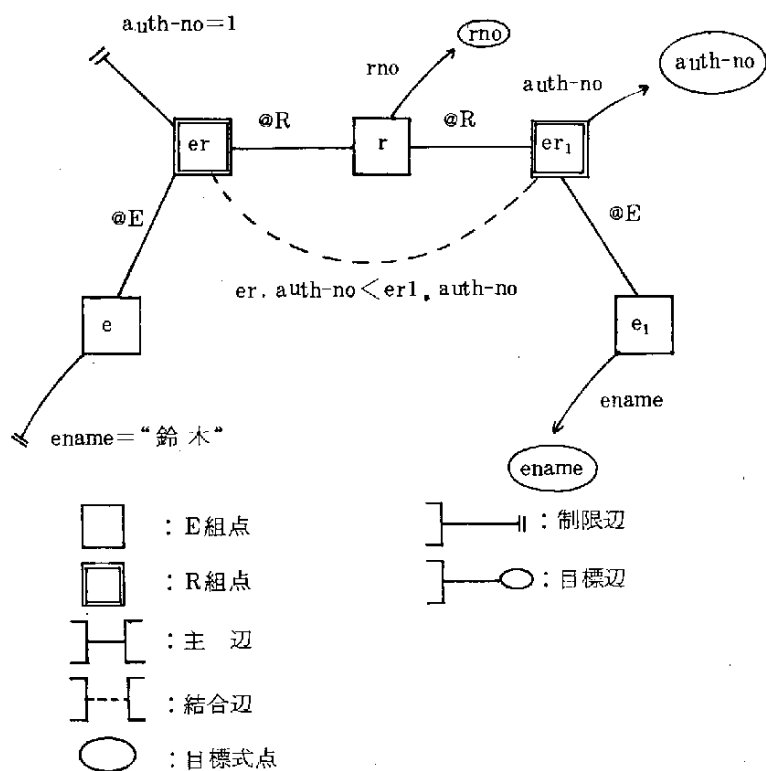


図 4.3 LQG の例

4.2.3 LC 操作言語 (LCML) — RQL 更新

次に、RQL の更新演算の構文と、その意味を RQL 検索と、LC 基本演算とによって表すことを試みる。以下、記号 x 及び e_i は、各々、関係スキーム X と E 関係スキーム E_i を表す組変数とする。

A 消去

関係 X に対する消去は、次の様に書ける。

```

range (x, X) (x1, X1) ... (xl, Xl); (l ≥ 0)
delete x where qual;

```

条件式 $qual$ を満足する X 内の組 t を全て消去する。

〔意味〕

```

T ← { t | t ∈ X ∧ qual (t, t1, ..., tl) ∧
      (t, t1, ..., tl) ∈ X × X1 × ... × Xl };
for ∀ t ∈ T do del (t, X);

```

〔例〕

図 4.1 の LC スキーマに対して考える。

(1) 「Jipdec を消去せよ」は、RQL で以下の様に表示せる。

```
range ( o, ORG );  
delete o where o. oname = " Jipdec " ;
```

これによって、Jipdec の全てのメンバ、及び、彼等とプロジェクトと論文の関係とか同時に消去される。

(2) 「鈴木がリーダーであるプロジェクトから、彼を除け」

```
range ( e, EMP ) ( pe, PEL );  
delete pe where  
e. ename = " 鈴木 " and e. @E = pe. @E ;
```

B 追 加

(1) E 関係 E への追加

```
range ( e1, E1 ) ... ( en, En ) ( x1, X1 ) ... ( xl, Xl );  
append to E ( a1 = exp1, ..., am = expm )  
where qual ;
```

ここで、 $\mathcal{Q}_E = \{ a_1, \dots, a_m \}$ 、又 $T_S(E) = \{ (\underline{B}_i(@E_i, @E), \underline{E}_i) \mid i = 1, \dots, n \}$ とする。

〔意味〕

```
T ← { t = ( e1, ..., en, e ) |  
  ( e1, ..., en, x1, ..., xl ) ∈ E1 × ... × En × X1 × ... × Xl ∧  
  qual ( e1, ..., en, x1, ..., xl ) ∧  
  e' = t1 ( e1, ..., en, x1, ..., xl ) } ;  
for ∀ ( e1, ..., en, e ) ∈ T  
  do app ( e, E | e1 ∈ E1, ..., en ∈ En ) ;
```

ここで、 $t_1 = (exp_1, \dots, exp_m)$ である。

E に qual を満たす組 e と共に、各 T 関係 B_i に、 $\otimes E_i(b_i) = \otimes E_i(e_i) \wedge \otimes E(b_i) = \otimes E(e)$ なる組 b_i を加える。

[例]

「鈴木を従業員とする(但し、田中と同じ会社とする)」

```

range (o, ORG)(e, EMP)(oe, OE);
append to EMP(eno = 5, ename = "鈴木", ... )
where   e. @E = oe. @E and
        e. ename = "田中" and oe. @O = o. @O ;

```

(2) P 関係 B への追加

B のスキームを $B(\otimes E_1, \otimes E_2)$ とする。

```

range (e1, E1)(e2, E2)(x1, X1) ... (xl, Xl);
append to B( @E1 = e1. @E1, @E2 = e2. @E2 )
where      qual ;

```

[意味] .

```

T ← { (e1, e2) |
    t = (e1, ..., en, x1, ..., xl) ∈ E1 × ... × En × X1 × ... × Xl ∧
    qual(t) }
for ∀ (e1, e2) ∈ T
do   app ( ( @E1(e1), @E2(e2) ), B | e1 ∈ E1, e2 ∈ E2 );

```

[例]

「DDBS プロジェクトのプロダクトとして LDP を加えよ」

```

range (d, PROD)(p, PROJ);
append to PD( @D = d. @D, @P = p. @P )
where
    d. dname = "LDP" and p. pno = "DDBS" ;

```

(3) G関係Gへの追加

Gのスキームを $\underline{G}(\ @E_1, \dots, @E_n, a_1, \dots, a_m)$ とする。

range (e_1, E_1) $\dots$ (e_n, E_n) (x_1, X_1) $\dots$ (x_l, X_l) ;

append to $\underline{G} (@E_1 = e_1, @E_1, \dots, @E_n = e_n, @E_n, \\ a_1 = \text{exp}_1, \dots, a_m = \text{exp}_m)$

where qual ;

[意味]

$T \leftarrow \{ (e_1, \dots, e_n, g) \mid$
 $t = (e_1, \dots, e_n, x_1, \dots, x_n) \in E_1 \times \dots \times E_n \times X_1 \times \dots \times X_l$
 $\wedge \text{qual}(t) \wedge g = \text{tl}(t) \} ;$
for $\forall (e_1, \dots, e_n, g) \in T$
 do app (($@E_1 (e_1), \dots, @E_n (e_n), g$) |
 $e_1 \in E_1, \dots, e_n \in E_n$) ;

ここで、 $\text{tl} = (\text{exp}_1, \dots, \text{exp}_m)$ である。

[例]

「鈴木を、田中がメンバであるプロジェクトのリーダーとする。」

range (e, EMP) (e_1, EMP) ($pe1, \text{PEL}$) (p, PROJ) ;
append to $\text{PEL} (@P = p, @P, @E = e, @E, \text{position} = \text{“リーダー”})$
where
 $e. \text{ename} = \text{“鈴木”}$ and
 $e1. \text{ename} = \text{“田中”}$ and $e1. @E = pe1. @E$ and
 $pe1. \text{position} = \text{“メンバ”}$ and $pe1. @P = p. @P ;$

C 置 換

関係Xの非主キー ($b_1, \dots, b_k$) ($b_i \in \Omega_X$) 値の置換は、次の様に書く。

range (x, X) (x_1, X_1) $\dots$ (x_n, X_n) ;
replace $x (b_1 = \text{exp}_1, \dots, b_k = \text{exp}_k)$
where qual ;

[意味]

```
T ← { ( x, x' ) |  
      t = ( x, x1, ..., xn ) ∈ X × X1 × ... × Xn ∧  
      qual ( t ) ∧ x' = t1 ( t ) } ;  
for ∀ ( x, x' ) ∈ T do rep ( x, x', X ) ;
```

ここで、 $t_1 = \{ \text{exp}_1, \dots, \text{exp}_k \}$ である。

[例]

(1) 論文番号“A”のコストを、論文“B”と同じにする。

```
range ( r, REP ) ( r1, REP ) ;  
replace r ( cost = r1. cost )  
where  
      r. rno = “A” and r1. rno = “B” ;
```

(2) 鈴木の所属を、田中と同じにする。

```
range ( e, EMP ) ( o, ORG ) ( oe, OE ) ( e1, EMP ) ( oe1, OE ) ;  
replace oe ( @O = o. @O )  
where e. ename = “鈴木” and  
      e. @E = oe. @E and  
      e1. ename = “田中” and e1. @E = oe1. @E and  
      oe1. @O = o. @O ;
```

5. LCMLのCQL表現

本章では、ローカル概念(LC)データ構造上の操作演算RQLを、論理CODASYLデータ構造上の(QLによる)CODASYL問合せで表せること示す。〔図5.1〕ここで、RQLによるLC問合せQを次の規則によって、CODASYL問合せQで表せる。

〔LC問合せからCODASYL問合せへの変換規則〕

- (1) LQ内の組変数の中で、E関係Eを表すeを、Eに1対1対応するレコード実現値(RO)集合Rを表すO変数rとする。eについての目標属性集合 tl_e と制限式 rf_e を各々rについての目標項目集合 tl_r と制限式 rf_r とする。各O変数の制限式の積を rf とし、目標項目集合の和を tl とする。
- (2) B関係スキーム $B(\otimes E_1, \otimes E_2)$ 、E関係スキーム E_1, E_2 を表す組変数を各々、 b, e_1, e_2 とする。 B, E_1, E_2 に対応する各々セット型 $S=(R_1, R_2)$ 、レコード型 R_1, R_2 とする。 R_1, R_2 を表すO変数を各々 r_1, r_2 とする。この時、LQが結合式 $(e_1, \otimes E_1=b, \otimes E_1 \text{ and } e_2, \otimes E_2=b, \otimes E_2)$ を条件式の積の要素として持つならば、O変数 r_1 と r_2 間に、セット述語 $P_S(r_1, r_2)$ をもうける。これらのセット述語の積を pf とする。
- (3) G関係スキーム $G(\otimes E_1, \dots, \otimes E_n, b_1, \dots, b_n)$ とE関係スキーム $E_i (i=1, \dots, n)$ に対する組変数を g と e_i とする。 R_i を E_i に対応したコード型とし、 r_i をそのO変数とする($i=1, \dots, n$)。セット型 $S_i=(R_i, R)$ ($i=1, \dots, n$)と $R=(Q_R=\{b_1, \dots, b_n\})$ は、 G に対応したリンク型 L とする。 R_i と R のO変数を各々 r_i と r とする。この時、LQが、結合式 $(e_i, \otimes E_i=g, \otimes E_i)$ を条件式の積の要素として持つならば、O変数 r_i と g に対するセット述語 $P_{S_i}(r_i, r)$ を pf に接合する。組変数 g についての制限式 rf_g と目標属性集合 tl_g とを、各々O変数 r の制限式 rf_r と目標項目集合 tl_r とする。
- (4) LQは、任意の組変数間の結合式を条件式の積要素としてもてる。E関係スキーム E_i と E_j の組変数を各々 e_i と e_j とする。 E_i と E_j に対応するレコード型を各々 R_i と R_j とし、各々O変数 r_i と r_j で表されるとする。ここで、結合式 jf_{e_i, e_j} を $qual$ の積要素として持つとき、対応する結合式 jf_{r_i, r_j} をもうけ、この積を jf とする。

G関係スキーム $G(\otimes E_1, \dots, \otimes E_n, b_1, \dots, b_n)$ の組変数を g とし、 e_i と g 間に結合式 $jf_{e_i, g}$ をLQの $qual$ の積要素としてもつとき、これをO変数 r_i と r 間の結合式 $jf_{r_i, r}$ とし、 jf に接合する。ここで r は、 G に対応したリンクの為のレコード型 R のO変数である。□

表5.1に、上記変換規則をまとめる。例として、図4.3のLCS問合せグラフ(LQG)について考える。このLQGを、変換したCODASYL問合せ(CQG)は図3.5となる。

以上より、次の命題が成り立つ。

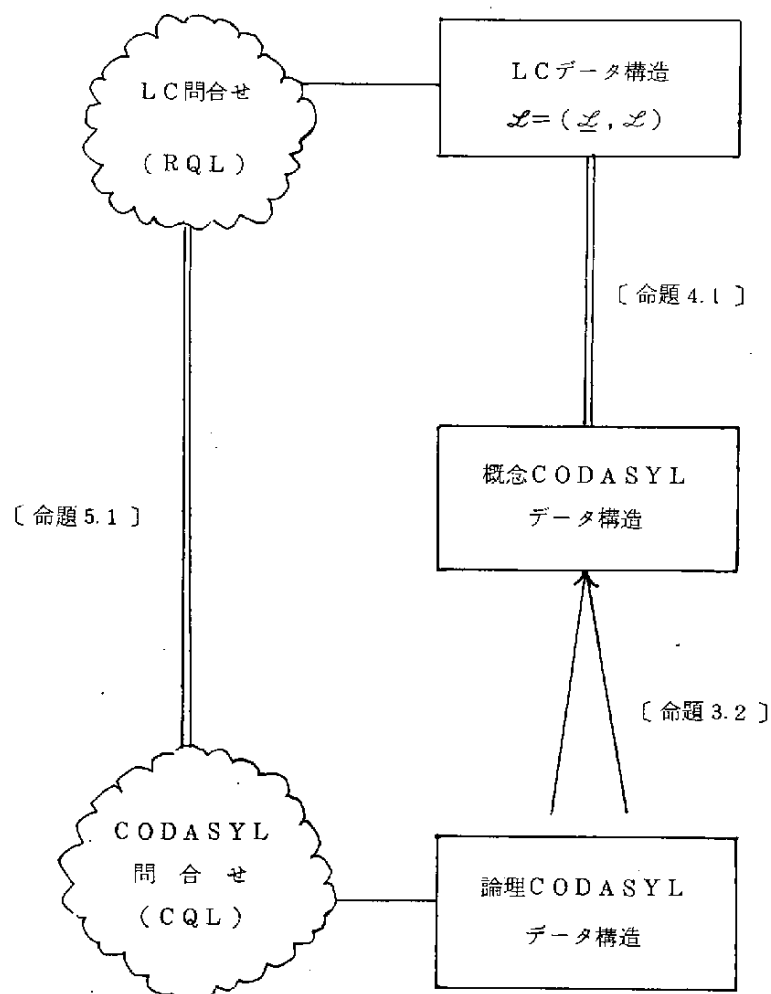
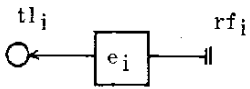
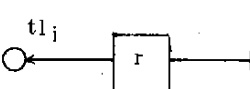
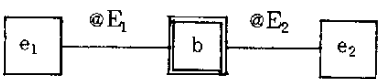
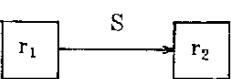
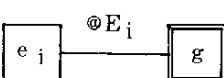
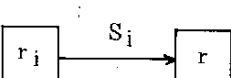
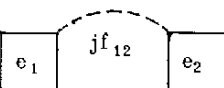
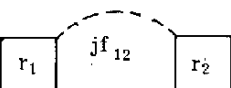
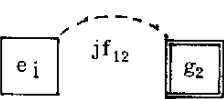
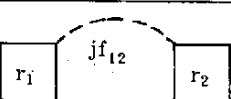
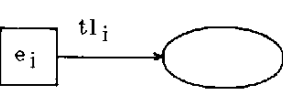
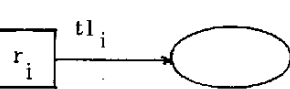
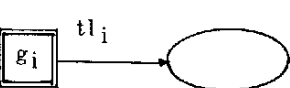
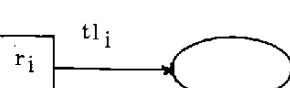


図 5.1 LC問合せとCODASYL問合せ

[命題 5.1]

LCデータ構造 $\mathcal{L}$ 上の全てのLC問合せを $\mathcal{L}$ と等価な論理CODASYLデータ構造 $\mathcal{C}$ 上のCODASYL問合せで表せる。 $\mathcal{L}$ と $\mathcal{C}$ が等価ならば、逆も成り立つ。

表 5.1 LQGとCQGとの対応

L Q G	C Q G
	
	
	
	
	
	
	

 : G組点

 : E組点

 : B組点

〔証明〕

命題 4.1 と 3.2 より、任意の LC データ構造 $\mathcal{L}$ と等価な論理 CODASYL データ構造 C が存在する。又、変換規則より、 $\mathcal{L}$ 上の任意の LC 問合せ Q に対して、 Q の結合式、制限式主結合式を各々 CODASYL 問合せ CQ の結合式、制限式、セット (S) 述語で表せる。又、 Q の各目標式を、 CQ の各目標式で表せる。よって、 Q に対して、同一の目標集合を導出する CQ が存在する。逆も、同様に成り立つ。■

6. CODASYL問合せのDML表現

本章では、論理データベースCに対してCQLで書かれたCODASYL問合せ(CQ)を仮想データベースV上のDMLで表せることと、その表現方法を論じる。命題5.1より、CODASYL問合せQは、CODASYL問合せグラフ(CQG)Gによって表せるので、ここではCQGを考える。

6.1 CQGのDMLによる表現

ここで、記号 r と r_i を各々レコード型 R と R_i を表す実現値(O)変数とする。 X と X_i を各々、 R と R_i に対応する仮想レコード(VR)型とする。 $Y=(X_1, X_2)$ と $Y_i=(X_{i1}, X_{i2})$ を各々、セット型 $S=(R_1, R_2)$ 、 $S_i=(R_{i1}, R_{i2})$ に対応した仮想セット(VS)型とする。

〔定理6.1〕

全てのCODASYL問合せを、DMLで表せる。

〔証明〕

- (1) 図6.1の様に、1つのO変数 r について、目標項目集合 tl_r と、制限式 rf_r が存在する場合について考える。このCQGの意味は、目標集合 $T=\{t \mid t=tl_r(x) \wedge x \in X \wedge rf_r(x)\}$ を導出することである。方法としては、次の3種がある

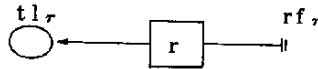


図6.1 CQG

- (i) 仮想レコード実現値(VRO)集合 X を順序関係 $\prec_X$ に基づいて全面サーチをする。全面サーチをしながら、条件 rf_r を満たす仮想レコード(VR)実現値 x の項目集合 tl_r の値の組を、出力テープ T_O に書けばよい。この為のDMLは、図6.2のようになる。
- (ii) r の制限式 rf_r が次の形式をもつとする。

$$rf_r ::= r.t=v \wedge rf_r'$$

ここで、 t は、VR型 X のCALC項目であり、 v は定数とする。 rf_r' は、 t 以外の項目についての制限式である。

このとき、まず作業域 β_X に v の値を格納してヘッドの連想機能を用いて、 $t(x)=v$ なるVR実現値 $x \in X$ に直接アクセスする(AC)ことができる。このDMLを、図6.3に示す。

```

open T.
find first X within A.*) go to L1.
M1. find next X within A.*)
L1. if SR=F go to E1.
get X. if ~rf, go to M1.
move tl, in X to tl, in T.
write T. go to M1.
E1. close T.

```

*) Aは、Xの属するエリア名¹⁾

図 6.2 図 6.1 の DML 表現

```

open T.
move v to t in X.
find any X. if SR=F go to E1.
get X. if ~rf, go to E1.
move tl, in X to tl, in T.
write T.
E1. close T.

```

図 6.3 図 6.1 の CALC を用いた DML

(iii) Xを、 $\langle X$ 以外の全順序関係 $\langle Y$ を用いて全面サーチする。ここで、VS型 $Y=(X_0, X_Y)$ とする(Y は、単項VS型である)。このときのDMLは、図 6.4 の様になる。

```

open T.
find first X within Y. go to L1.
M1. find next X within Y.
L1. if SR=F go to E1.
get X. if ~rf, go to M1.
move tl, in X to tl, in T.
write T. go to M1.
E1. close T.

```

ここで Y は、 $Y=(X_0, X)$ なる単項VS型である。

図 6.4 図 6.1 の単項VS O集合を用いたDML

(2) 図 6.5 の2つの0変数 r_1 と r_2 から成るCQGについて考える。このCQGは、目標集合

$$T = \{ t \mid t = (tl_1(x_1), tl_2(x_2)) \wedge x_1 \in X_1 \\ \wedge x_2 \in X_2 \wedge P_Y(x_1, x_2) \wedge \\ rf_1(x_1) \wedge rf_2(x_2) \wedge jf_{12}(x_1, x_2) \}$$

を導出することである。即ち、VRO集合 X_1 と X_2 の中で、VR実現値 $x_1 \in X_1$ と $x_2 \in X_2$ が親子関係にあり、かつ各々が制限式 rf_1 と rf_2 を満足する x_1 と x_2 の対の中で結合式 jf_{12} を満たすとき、目標項目 tl_1 と tl_2 の値の組を出力テープ T_0 に書き出すことである。この方法としては (i) まず、 X_1 内の rf_1 を満たす $x_1 \in X_1$ を見つけ、その Y の子 $x_2 \in X_2$ を順序関係 $\langle_Y$ に基づいてアクセスする方法と、(ii) 逆に $x_2 \in X_2$ を見つけ、その Y の親 $x_1 \in X_1$ をアクセスする方法との2つが可能である。各々、 X_1 と X_2 のアクセス方法は、(i) の3種の方法が可能であるが、便宜上ここでは、(i) の全面サーチ方法を用いている。(i) のDMLを図 6.6 に(ii) のDMLを図 6.7 に示す。

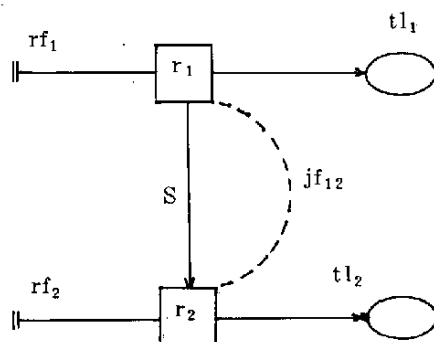
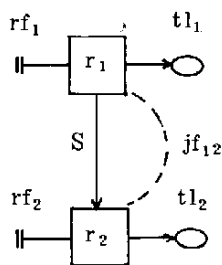


図 6.5 CQG の例

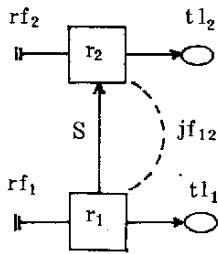


```

open T.
find first X1 within A. go to L1.
M1. find next X1 within A.
L1. if SR=F go to E1.
    get X1. if ~rf1 go to M1.
    move tl1 in X1 to tl1 in T.
    find first X2 within S. go to L2.
M2. find next X2 within S.
L2. if SR=F go to M1.
    get X2. if ~(rf2 ∧ jf12) go to M2.
    move tl2 in X2 to tl2 in T.
    write T. go to M2.
E1. close T.

```

図 6.6 図 6.5 のDML(i)



```

open T.
find first X2 within A. go to L2.
M2. find next X2 within A.
L2. if SR=F go to E1.
    get X2. if ~ rf2 go to M2.
    move tl2 in X2 to tl2 in T.
    find owner within Y. if SR=G go to M2.
    get X1. if ~(rf1 ∧ jf12) go to M2.
    move tl1 in X2 to tl1 in T.
    write T. go to M2.
E1. close T.

```

図 6.7 図 6.5 の DML(II)

(III) 次に、一般の主辺で連結な n (≤ 3) 個の点から成る CQG について考える。CQG の内の各主辺 (r_i, r_j) と、点 r_i と r_j から成る部分 CQG C_{ij} は、(II) と同じ形式であるので、(II) より目標集合 T_{ij} を導出する DML で表せる。ここで、各 r_i と r_j の目標項目として、各々の db キーと、 r_i と r_j 各々他の点間の結合式内の項目とを加えて、目標集合 T_{ij} を導出する。各主辺から得られた中間目標集合 T_{ij} を、db キーについて等結合を行い、結合辺の結合式の条件を満たすものを、問合せの目標項目で射影した組の集合を目標集合とすればよい。

(i)、(ii)、(iii) より、全ての CQG を、DML で表せることが解かる。■

[定義 6.1]

CQG の DML 表現を、DML プログラムとする。□

6.2 出力テープの最適化

定理 1 の (iii) の方法では、CQG の主辺数を n とすると、 n 本の中間目標用出力テープと、1 本の目標集合 (T) 出力用テープとの都合 $n+1$ 本の出力テープと、 T を生成する為に n 本のテープのマージ操作が必要になる。1 本の出力テープは、実際の計算機では、1 つのファイルに対応している。よって、出力テープ数を減少し、出力テープ操作数を減少させることは、ファイル空間とファイル操作とを減少させることに対応するので、処理効率上望ましい。

[定義]

ここで、次の必要十分条件を満足する DML プログラム P を、巡航的 (navigational) であるという。

- (i) 1本の出力テープ(T_0)のみを用いて
- (ii) T_0 への`open`と`close`演算は、P内で各々1回だけ用いられ
- (iii) これら以外の T_0 への演算としては、`write`演算だけが用いられる。

即ち、DMLプログラムPが、順次ファイル(SAMファイル)Tに対して、順々に書き出している、書き終り`close`演算を行った時点で、T内に目標集合がつけられるときに、Pを巡航的なDMLプログラムという。□

〔定義 6.2〕

CODASYL問合せグラフ(CQG)Gが、主辺について連結で、ある点を根の点とし主辺について閉路が存在しないとき、Gは木型であるとする。この木型のG内の点の主辺についての次数が1のものを葉の点とする。根でも葉でもない点を中間点とするG内の根と葉との間の主辺で連結な点と主辺から成る経路を、上下路とする。G内の各主辺において、主辺Pと接する2点のうち、根に近い点xをpの父、他の点yをpの息子とする。又、主辺pは、S型名Sをラベルとして持ち、pをS(x, y)と表す。S型 $S = (R_i, R_j)$ を表す主辺S(x, y)において、yが R_j を表す点のときyを子(M)型、逆にyが R_i を表すときにはyを親(O)型とする。□

例えば、図6.8は、点aを根とする木型CQGの例である。点d、e、f、h、j、k、lは葉であり、b、c、g、iは中間点である。

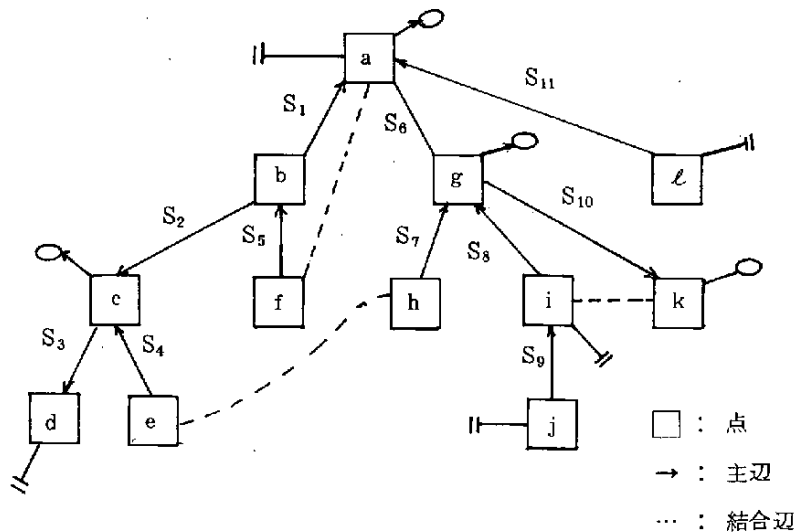


図 6.8 木型CQGの例

主辺 S_1 に対して、点bはO型である。点kは S_{10} に対してM型である。経路a、 S_6 、g、 S_8 、i、 S_9 、jは、上下路である。

〔定義 6.3〕

CODASYL 問合せグラフ (CQG) G において、目標辺に接する点を、目標点とする。結合辺に接する点を結合点とする。 G 内の点が、目標点又は結合点のときにこれを出力点とする。又、制限辺に接する点を制限点とする。□

〔定義 6.4〕

$G_1 = (N_1, E_1)$ と $G_2 = (N_2, E_2)$ を CODASYL 問合せグラフとする。このとき、次の必要十分条件が成り立つとき、 G_1 と G_2 は等価であるとする。

- (1) G_1 と G_2 との目標項目集合が同一である。
- (2) G_1 と G_2 との各々の制限辺、結合辺、主辺の表す制限式、結合式、親子述語の積が等価な論理式である。

このとき、 G_1 と G_2 とは、互いに他を正しく表すという。□

〔定義 6.5〕

CODASYL 問合せグラフ (CQG) G が、以下の必要十分条件を満足するとき、 G は巡航木である。

- (i) G は木であり各点を父とする息子が順序づけられている。
- (ii) 全ての目標点が、 G 内の最右の上下路内にある。
- (iii) 各結合辺に接する 2 点は、同一の上下路内にある。

このとき、根と、一番葉に近い目標点との間の路を幹とする。幹は全ての目標点を含む。 G の各点 r に対して、 r より根に近い点 r_i との結合辺 (rf_{rr_i}) の集合の表す結合式 r の上方の結合式 uf_r (rf_{rr_i} の積) とし、 r より葉に近い点 r_j との結合辺 (rf_{rr_j}) の集合の表す結合式を r の下方の結合式 df_r (rf_{rr_j} の積) とする。 r から根までの径路円。点の集合を $r_0, \dots, r_j, r$ とすると、 $uf_r : \text{dom}(\mathcal{Q}_{r_0}) \times \dots \times \text{dom}(\mathcal{Q}_{r_j}) \wedge \text{dom}(\mathcal{Q}_R) \rightarrow \{\text{true}, \text{false}\}$ である。□

図 6.8 の木は、(ii) 及び (iii) に反するので巡航木ではない。図 6.9 の CQG は、定義の条件を満たすので巡航木である。点で、 $uf_f = jf_{af} \wedge jf_{fd}$ であり、 $df_f = jf_{fg}$ である。

〔定義 6.6〕

CODASYL 問合せグラフ (CQG) G が、巡航木でかつ、全ての点の主辺についての次数が 2 以下の時、 G を線型とする。□

〔定義 6.7〕

G を、木型の CQG とする。 G 内の各主辺 $S(r_i, r_j)$ に対して、述語 $B_S : \text{dom}(\mathcal{Q}_{r_i}) \times$

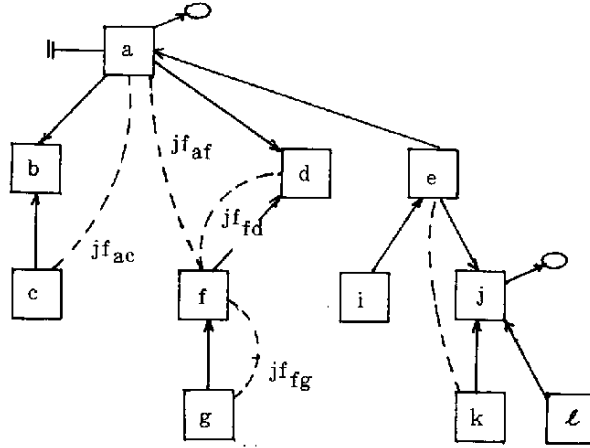


図 6.9 巡航木

$\text{dom}(\mathcal{Q}_{R_j}) \rightarrow \{\text{true}, \text{false}\}$ を定める。

$$B_S(x_i, x_j) = \begin{cases} \text{true} & \text{if } x_i \in X_i \wedge x_j \in X_j \wedge \\ & ((r_j \text{ は M 型} \wedge P_S(x_i, x_j)) \vee \\ & (r_j \text{ は O 型} \wedge P_S(x_j, x_i))) \\ \text{false} & \text{otherwise} \end{cases}$$

□

〔定義 6.8〕

G を任意の巡航木とし、点 r を G 内の点とし、 r_i は、息子 $r_{i1}, \dots, r_{in}$ ($n \geq 0$) を各々主辺 $S_{i1}, \dots, S_{in}$ について持ち、又、 r_i は主辺 S_i の息子とする。 G の根を r_0 とし、 r_0 と r 間の径路を $r_0, S_1, r_2, \dots, r_{i-1}, S_i, r_i$ とする。この時、VRO 集合 X_i 内の VR 実現値 $x_i (\in X_i)$ は、次の条件を満たすとき、上方の条件を満足するという。

$$u\text{cond}_i(x_i) = \begin{cases} \text{true} & \text{if } \bigwedge_{j=0}^i [x_j \in X_j \wedge rf_j(x_j)] \\ & \bigwedge_{j=1}^i [B_{S_j}(x_{j-1}, x_j) \wedge \\ & \quad uf_j(x_0, \dots, x_{j-1}, x_j)] \\ & (\text{i.e. } \exists x_{i-1} \in X_{i-1} (u\text{cond}_{i-1}(x_{i-1}) \wedge \\ & \quad B_{S_i}(x_{i-1}, x_i) \wedge rf_i(x_i))) \\ \text{false} & \text{otherwise} \end{cases}$$

即ち、 x_i が $ucond_i(x_i) = \text{true}$ の時、 r_0 と r_i 間の径路内に、各点の制限式、各主辺のS述語、径路内の点間の結合式を満たす1組のVR実現値 $(x_0, x_1, \dots, x_{i-1}, x_i)$ ($x_j \in X_j$ $j=0, 1, \dots, i$) が存在することである。

一方、点 r_i の各息子 r_{ij} は部分木 T_{ij} の根であり、 r_i は、 G の部分木 T_i の根である〔図 6.10〕この時、VRO 集合 X_i 内の各VR実現値 $x_i \in X_i$ が、次の条件 $dcond_i(x_i)$ を満足する時、 x_i は下方条件を満足するとする。

$$dcond_i(x_i) \triangleq \begin{cases} \text{true} & \text{if } \forall r_{ij} (j=1, \dots, n_i) \text{ に対して、} \\ & \exists x_{ij} \in X_{ij} \\ & (B_{S_{ij}}(x_i, x_j) \wedge rf_j(x_j) \wedge \\ & \quad uf_j(x_0, \dots, x_i, x_j) \wedge \\ & \quad dcond_{ij}(x_j)) \\ \text{false} & \text{otherwise.} \end{cases}$$

即ち、点 r_i を通る上下路を持つ各葉点（即ち、部分木 $T_{i1}, \dots, T_{in_i}$ 内の全ての葉点）を l_i とする。 r_i から l_i の径路に対して、 l_i の表すVRO集合 L_i 内に、高々1つの $ucond_{l_i}(u_i \in L_i)$ を満たすVR実現値 u_i が存在するならば、 x_i は $dcond_i(x_i)$ を満たす。

各VR実現値 $x_i \in X_i$ が条件を満たすとは、 $ucond_i(x_i) \wedge dcond_i(x_i) = \text{true}$ のときである。□

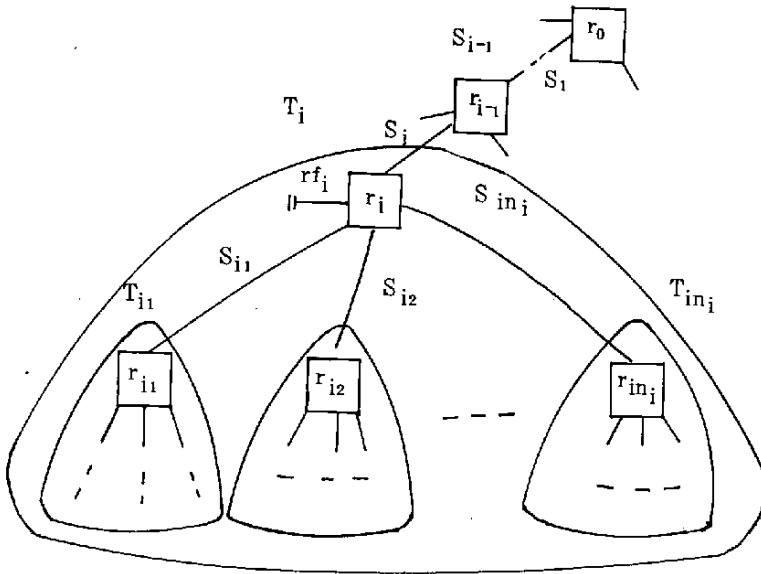


図 6.10 G の部分木 T_i

〔定義 6.9〕

T を任意の巡航木とする。 T の幹を $r_0, S_1, r_1, S_2, \dots, S_k, r_k$ と、幹内の各点 r_i が図 6.1.1 の様に、 r_i の幹外の息子 r_{ij} を根とする部分木 T_{ij} ($j=1, \dots, l_i, l_i \geq 0$) を持つとする。このとき、部分木 $T_{i1}, \dots, T_{il_i}$ を持つことになる。□

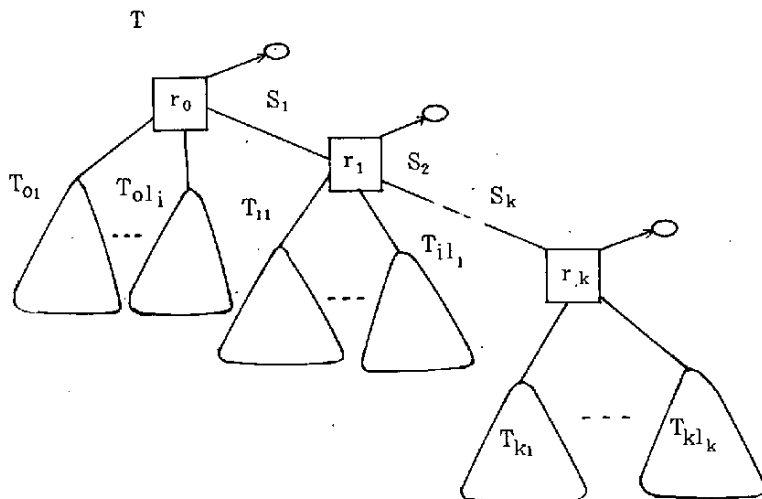


図 6.1.1 巡航木 T

〔定義 6.1.0〕

T_{ij} の全ての葉点を $r_{ij1}, \dots, r_{ijl_{ij}}$ とする補助定理 6.2.2 より、枝 T_{ij} が条件をみたすとは、 r_i の上方条件 $ucond_i(x_i) = \text{true}$ なる VR 実現値 $x_i \in X_i$ に対して、全ての葉点 r_{ijk} が、少なくとも 1 つの r_{ij} 条件を満たす VR 実現値 $x_{ijk} \in X_{ijk}$ ($ucond_{ijk}(x_{ijk}) = \text{true} \wedge dcond_{ijk}(x_{ijk}) = \text{true}$) を持つことである。各 $x_i \in X_i$ に対して、 r_i から根 r_0 までの径路内の条件を満たす VR 実現値の組を $(x_0, x_1, \dots, x_i)$ とする。えのとき、各 $x_i \in X_i$ に対して、

$$cond_{T_{ij}}(x_i) = \begin{cases} \text{true} & \text{if } Bs_{ij}(x_i, x_{ij}) \wedge (x_i \text{ に対して、枝 } T_{ij} \text{ の条件を満たす}) \\ \text{false} & \text{otherwise} \end{cases} \quad \square$$

〔定義 6.1.1〕

巡航木 T 〔図 6.1.1〕の意味は、次の様である。

$$T = \{ t \mid t = t_1(x_0, x_1, \dots, x_k) \wedge x_0 \in X_0 \wedge x_1 \in X_1 \wedge \dots \wedge x_k \in X_k \wedge$$

$$\wedge \text{rf}_0(x_0)$$

$$\bigwedge_{i=1}^k [\text{rf}_i(x_i) \wedge \text{uf}_i(x_0, x_1, \dots, x_k)]$$

$$\bigwedge_{j=1}^{l_i} \text{cond}_{r_{ij}}(x_i)$$

$$\wedge B_{S_i}(x_{i-1}, x_i) \} \} \quad \square$$

〔補助定理 6.2.1〕

全ての線型のCODASYL問合せグラフ(CQG)を、DMLで巡航的に表せる。

〔証明〕

図 6.1.2 の様な点 r_0 を根とする線型の CQG G を考える。 G の意味は、次の目標集合 T を導出することである。

$$T = \{ t \mid t = (t_{l_0}(x_0), t_{l_1}(x_1), \dots, t_{l_k}(x_k)) \}$$

$$\bigwedge_{i=0}^k [x_i \in X_i \wedge \text{rf}_i(x_i)]$$

$$\bigwedge_{i=1}^k B_{S_i}(x_{i-1}, x_i)$$

$$\bigwedge_{i=1}^k \bigwedge_{j=0}^{i-1} \text{if}_{r_i r_j}(x_i, x_j) \}$$

この T は、次の様にして、仮想データ構造から導出できる。

- (i) X_0 の中から、順に rf_0 を満たす VR 実現値 $x_0 (\in X_0)$ を見つける。
- (ii) 各 $i = 1, \dots, k$ に対して、条件を満たすある $x_{i-1} \in X_{i-1}$ に対して、条件 $B_{S_i}(x_{i-1}, x_i)$ と $\text{rf}_i(x_i)$ とを共に真とし、かつ、結合式 $\text{uf}_i(x_0, x_1, \dots, x_i)$ を満たすある $x_i \in X_i$ を VS 型 Y_i の順序関係に基づいて見つける。1 つ見つかったならば、次の $i (\leftarrow i + 1)$ へ。
- (iii) 見つからなければ、 $i \leftarrow i - 1$
- (iv) $i = k$ で、(ii) の $x_k \in X_k$ がみつかった時、組 $(x_0, x_1, \dots, x_k)$ の中で、項目集合 $\{t_{l_0}, t_{l_1}, \dots, t_{l_k}\}$ の値を、 T の 1 つの組 t として出力する。 $i \leftarrow i - 1$ として、(ii) へ。
- (i) ~ (iv) を DML で表せることは表 3.2 の DML の意味より自明である。よって、定理は成り立つ。■

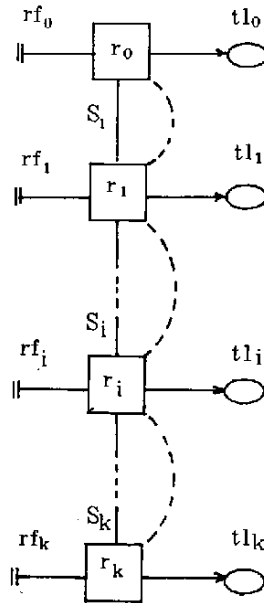


図 6.1 2 r_0 を根とする線型 C Q G

〔補助定理 6.2.2〕

線型 C Q G に対して、この目標集合内のある 1 つの組だけを求める巡航的な D M L が存在する。

〔証明〕

補助定 6.2.1 の (V) において、1 つでも $x_h \in X_h$ が見つかった時に、組 $(x_0, x_1, \dots, x_h)$ を出力して終了すればよい。■

補助定理 6.1 は、条件を満足する全て (V) の組を見つけるのに対して、補助定理 6.2 は、条件を満足するある (exists) 組を見つけることに対応する。例えば、例として、図 3.3 の CODASYL スキーマに対して、問合せとして「Jipdec の全ての従業員を求めよ」は、図 6.6 より次の様な D M L となる。

```

open T.
  find first ORG within A. go to L1.
M1.  find next ORG within A.
L1.  if SR=F go to E1.
      get ORG.
      if not (oname in ORG="Jipdec") go to M1.
      find first EMP within OE. go to L2.
  
```

```

M2.  find next EMP within OE.
L2.  if SR=F go to M1.
      get EMP.
      move ename in EMP to ename in T.
      write T. go to M2.
E1.  close T.

```

一方、「Jipdec のある職員を求めよ」に対するPMLは、次の様になる。

```

      open T.
      find first ORG within A. go to L1.
M1.  find next ORG within A.
L1.  if SR=F go to E1.
      get ORG.
      if not ( oname in ORG="Jipdec" ) go to M1.
      find first EMP within OE. go to L2.
M2.  find next EMP within OE.
L2.  if SR=F go to M1.
      get EMP.
      move ename in EMP to ename in T.
      write T.
E1.  close T.

```

〔定理6.2〕

全ての巡航木型のCODASYL問合せグラフ(CQG)を、DMLで巡航的に表せる。

〔証明〕

Tを図6.11の様な任意の巡航木とする。この巡航木Tの意味は、定義6.11に示したものである。各幹点 r_i の各子の部分木 T_{ij} が条件を満たすかどうかを明らかにするDMLが存在することは、補助定理6.2.2より明らかである。又、幹内の点から、条件を満たす全てのVR実現値を求めるDMLが存在することも、補助定理6.2.1より明らかである。よって、定理は成り立つ。■

〔定理6.3〕

任意のCQGを巡航木で表せる。

〔証明〕

任意のCQGをGとする。線型CQGは巡航木であるから、Gを線型CQGで表せることを示せば

十分である。任意のグラフの点と主辺を複数回訪ねることを許して、1筆書きで訪ねることができるので、任意のCQG G の線型CQGで表せる。ここで、 G を、上記の様に訪ねて出来た線型CQGを T とする。このとき、 G 内のある点 x は、 T 内に複数回現われる。これを、根に近い方から順序づけて、 $x_1, \dots, x_n$ ($n \geq 2$) であるとする。 T と G の意味を同一にするには、

- (1) x の制限辺 rf_x 、目標辺 tl_x を、 x_1 にだけ付ける。 $x_2, \dots, x_n$ には付加しない。
- (2) x の結合辺 uf_x, df_x も、 x_1 にだけ付ける。
- (3) T に新たに、結合辺 $x_1, \odot X = x_i, \odot X$ ($i = 2, \dots, n$) を、 x_1 と各 x_i 間に設ける。

以上によって、 G と T は、同一の目標項目集合と、同一の制限式、論理的に等価な結合式とセット式をもつことになるので等価であるといえる。■

[定義 6.1 2]

定理 4.3 で、 G 内のある主辺 S が、複数回訪ねられ、 T 内に、訪問順に $S_1, \dots, S_n$ ($n \geq 2$) が存在するとする。この時、2 回目の訪問以降の $S_2, \dots, S_n$ の息子の点を B (後退) 型とする。□

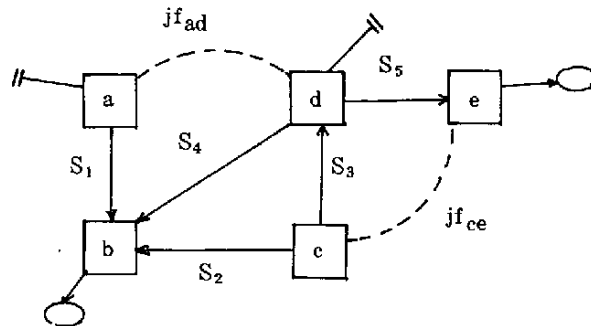


図 6.1 3 CQG の例

例として、図 6.13 の CQG G を考えてみよう。この時、 G の点 a から始めて、 $a, S_1, b, S_2, c, S_3, d, S_4, e, S_5, e, S_4, b$ と訪ねたとすると、図 6.1 4 の線型 CQG となる。図 6.1 4 からわかる様に、点 b と d は 2 回訪ねられるので、 b_1 と b_2, d_1 と d_2 間に db キーの等結合辺が付加される。又 G の主辺 S_5 は 2 回訪ねられるので、 S_5 の息子点 d_2 は B 型となる。

図 6.2 を表す、図 6.1 5 の様な巡航木も存在する。

[定理 6.4]

全ての CQG を、DML で巡航的に表せる。

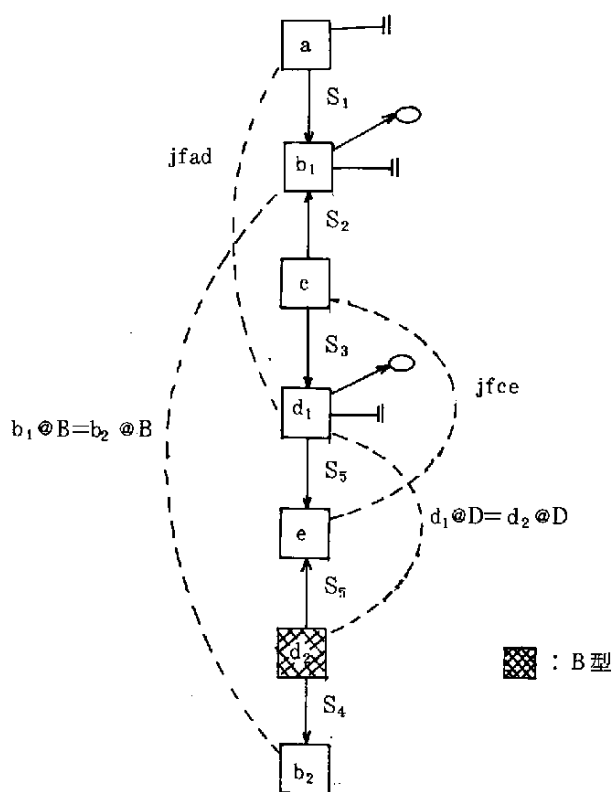


図 6.1 4 図 6.1 3 の線型 C Q G (巡航木)

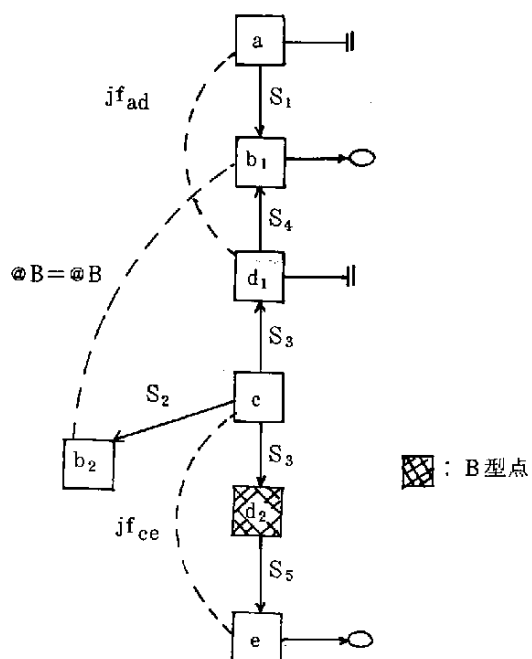


図 6.1 5 図 6.1 3 の C Q G の例

〔証明〕

定理 6.2、6.3 より明らかである。■

〔定理 6.5〕

全ての CQG を、仮想 CODASYL マシン (VCM) 上でシミュレートできる。

〔証明〕

命題 3.2 より全ての DML を、VCM 上でシミュレートできるので、定理 6.4 より明らかである。■

よって、任意の CQG を、巡航木で表し、この巡航木を、巡航的な DML プログラムで表せることが論理的に明らかになった〔図 6.16〕。次に、具体的な、巡航木から DML の変換アルゴリズムと、CQG から巡航木の変換アルゴリズムとを示す。

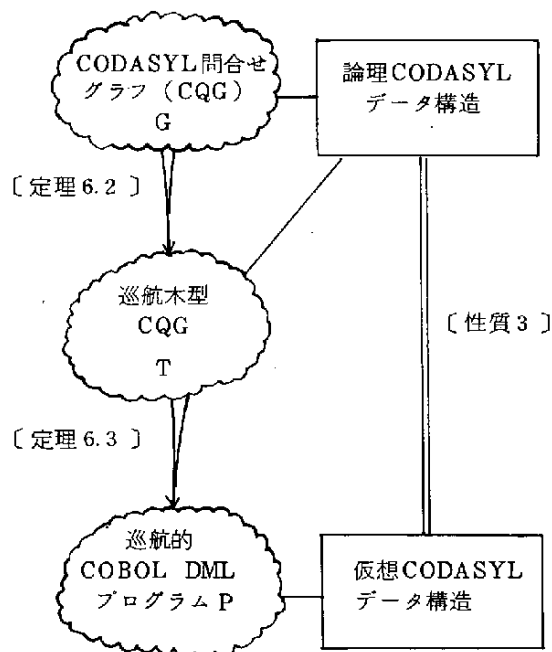


図 6.16 CQG と DML

6.3. DML 生成アルゴリズム

T を任意の巡航木とする。T 内の各点は、点の T 内での位置（根、中間、葉）、点の型（M、O、B）、及び幹内か幹外かの 3 点によって表 6.1 の様に 9 種に分類出来る。

表 6.1 巡航木内の点の分類

分 類	位 置	幹／幹外	型
根	根	幹	/
M型幹点	中 間	幹	M
O型幹点	中 間	幹	O
M 型 点	中 間	幹外	M
O 型 点	中 間	幹外	O
M型幹葉	葉	幹	M
O型幹葉	葉	幹	O
M 型 葉	葉	幹外	M
O 型 葉	葉	幹外	O

[DMLプログラム生成アルゴリズム (DMLG)]

入力 巡航木 $T = (N, E)$

出力 DMLプログラム P

手順

T を、根点から始めて、主辺 (枝) S とその子息点 r との対を、前置順 (preoder) に訪ねながら、各対に対して表 6.2 に基づいて DML を順々に、プログラムテープ T_P に出力する。

- (1) pushdown (A, A, A) $S \leftarrow A$; $r_P \leftarrow A$;
 $r \leftarrow T$ の根 ; DMLG (r_P, S, r) ;
 $i \leftarrow 0$; $r_P \leftarrow r$;
- (2) $i \leftarrow i + 1$;
 $r \leftarrow r_P$ の第 i 番目の息子 ;
 $S' \leftarrow r$ と r_P 間の主辺 ;
(S, r) が見つければ、pushdown (S, r_P, i) ;
 $S \leftarrow S'$; DMLG (r_P, S, r) ;
 $r_P \leftarrow r$; $i \leftarrow 0$;

go to (2) ;
- (3) (S', r') の見つからなかったとき、
popup (S, r, i) ;
if $S \neq A$ then go to (2) ;

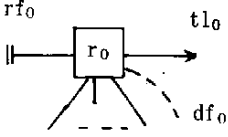
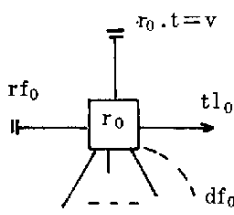
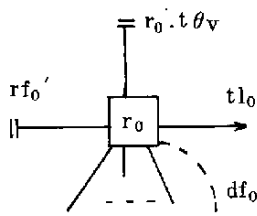
else
DMLG (A, A) ;

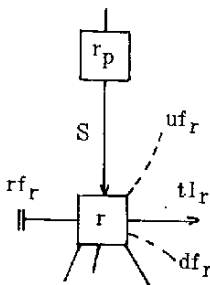
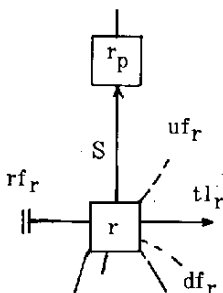
procedure DMLG(S, r);

(S, r)の対に対して、表6.2よりDMLをT_Pに書き出す。□

より詳細なCOBOL DMLのパターンについては、付記4を参照されたい。

表6.2 CQGとDML

r _P	S	r	C Q G	D M L
A	A	r ₀	a) 根 	<u>open</u> T. <u>move</u> o <u>to</u> B'. <u>find first</u> X ₀ <u>within</u> A. <u>go to</u> Lo. K ₀ . ([<u>find</u> X ₀ ; <u>db-key</u> <u>is</u> δ ₀ .]) M ₀ . <u>find next</u> X ₀ <u>within</u> A. L ₀ . <u>if</u> SR=F <u>go to</u> E ₀ <u>get</u> X ₀ . <u>if</u> ~ rf ₀ <u>go to</u> M ₀ . <u>accept</u> δ ₀ <u>from</u> X ₀ <u>current</u> . <u>move</u> o t ₀ <u>in</u> X ₀ <u>to</u> v ₀ <u>move</u> t ₀ <u>in</u> X ₀ <u>to</u> t ₀ <u>in</u> B
			b) 	<u>open</u> T. <u>move</u> o <u>to</u> B'. <u>move</u> v <u>to</u> t <u>in</u> X ₀ . <u>find any</u> X ₀ . <u>if</u> SR=F <u>go to</u> E ₀ . <u>get</u> X ₀ . <u>if</u> ~ rf ₀ <u>go to</u> E ₀ . <u>accept</u> δ ₀ <u>from</u> X ₀ <u>current</u> . <u>move</u> o t ₀ <u>in</u> X ₀ <u>to</u> v ₀ <u>move</u> t ₀ <u>in</u> X ₀ <u>to</u> t ₀ <u>in</u> B.
			c) 	<u>open</u> T. <u>move</u> o <u>to</u> B'. <u>find first</u> X ₀ <u>within</u> Y. <u>go to</u> Lo. K ₀ . ([<u>find</u> X ₀ ; <u>db-key</u> <u>is</u> δ ₀ .]) M ₀ . <u>find next</u> X ₀ <u>within</u> A. L ₀ . <u>if</u> SR=F <u>go to</u> E ₀ . <u>get</u> X ₀ . <u>accept</u> δ ₀ <u>from</u> X ₀ <u>current</u> . <u>if</u> t <u>in</u> X ₀ θ' v <u>go to</u> E ₀ . <u>if</u> ~ (rf ₀ ' <u>and</u> t <u>in</u> X ₀ θ v) <u>go to</u> M ₀ . <u>move</u> o t ₀ <u>in</u> X ₀ <u>to</u> v ₀ . <u>move</u> t ₀ <u>in</u> X ₀ <u>to</u> t ₀ <u>in</u> B.

r_p	S	r	C Q G	D M L
			θ θ' 昇 $=$ $<$ $>$ $\leq$ $\geq$ $<$ 降 $=$ $>$ $<$ $\geq$ $\leq$ $>$	ここで $\underline{Y} = (\underline{X}, \underline{X}_0)$ ($\underline{X}$ はシステムVR型) t は $\underline{Y}$ のソート項目
r_p	S	r	d) M型幹点 	Sr. $\llcorner$ <u>find</u> X_p ; <u>db-key is</u> δ_{r_p} . $\lrcorner$ <u>find first</u> X_r <u>within</u> Y . <u>go to</u> L_r . Kr. $\llcorner$ <u>find</u> X_r ; <u>db-key is</u> δ_r . $\lrcorner$ Mr. <u>find next</u> X_r <u>within</u> Y . Lr. <u>if</u> $SR=F$ <u>go to</u> K_L . <u>accept</u> δ_r <u>from</u> X_r <u>current</u> <u>get</u> X_r . <u>if</u> $\sim(rf_r \wedge uf_r)$ <u>go to</u> M_r . <u>move</u> ot_r <u>in</u> X_r <u>to</u> v_r . <u>move</u> tl_r <u>in</u> X_r <u>to</u> tl_r <u>in</u> B . $\llcorner$ <u>if</u> $B' \neq B$ $T \leftarrow tl(B)$ <u>write</u> T <u>move</u> B <u>to</u> B' . <u>go to</u> M_r . $\lrcorner$
r_p	S	r	e) O型幹点 	Sr. $\llcorner$ <u>find</u> X_p ; <u>db-key is</u> δ_{r_p} . $\lrcorner$ <u>find owner within</u> Y . <u>if</u> $SR=F$ <u>go to</u> K_L . <u>accept</u> δ_r <u>from</u> X_r <u>current</u> . <u>get</u> X_r . <u>if</u> $\sim(rf_r \wedge uf_r)$ <u>go to</u> K_L . <u>move</u> ot_r <u>in</u> X_r <u>to</u> v_r . <u>move</u> tl_r <u>in</u> X_r <u>to</u> tl_r <u>in</u> B . $\llcorner$ <u>if</u> $B' \neq B$ $T \leftarrow tl(B)$ <u>write</u> T . <u>move</u> B <u>to</u> B' . <u>go to</u> K_L . $\lrcorner$

r_p	S	r	C Q G	D M L
r_p	S	r	f) M型点 	Sr. $\ll$ <u>find</u> X_p ; <u>db-key</u> <u>is</u> δ_{rp} . $\gg$ <u>find first</u> X_r <u>within</u> Y . <u>go to</u> L_r . Kr. $\ll$ <u>find</u> X_r ; <u>db-key</u> <u>is</u> δ_r . $\gg$ Mr. <u>find next</u> X_r <u>within</u> Y . Lr. <u>if</u> $SR=F$ <u>go to</u> K_l . <u>accept</u> δ_r <u>from</u> X_r <u>current</u> . <u>get</u> X_r . <u>if</u> $\sim(rf_r \wedge uf_r)$ <u>go to</u> M_r . <u>move</u> ot_r <u>in</u> X_r <u>to</u> vr . (<u>if</u> $B' \neq B$ $T \leftarrow tl(B)$ <u>write</u> T <u>move</u> B <u>to</u> B' . <u>go to</u> M_r .) $\ll$ <u>go to</u> S_k . $\gg$
r_p	S	r	g) O型点 	Sr. $\ll$ <u>find</u> X_p ; <u>db-key</u> <u>is</u> δ_{rp} . $\gg$ <u>find owner</u> <u>within</u> Y . <u>if</u> $SR=F$ <u>go to</u> K_l . <u>accept</u> δ_r <u>from</u> X_r <u>current</u> . <u>get</u> X_r . <u>if</u> $\sim(rf_r \wedge uf_r)$ <u>go to</u> K_l . <u>move</u> ot_r <u>in</u> X_r <u>to</u> vr . (<u>if</u> $B' \neq B$ $T \leftarrow tl(B)$ <u>write</u> T . <u>move</u> B <u>to</u> B' . <u>go to</u> K_l .) $\ll$ <u>go to</u> S_k . $\gg$
				Eo. <u>close</u> T .

表 6.2 で、各点に対して、

$r_l =$ r から根点までの経路内で、 r に最も近い M 型点。

M 型点の無いとき、 r_l は根点

$r_k =$ r_p を親とする r の右隣りの点、なければ r_l となる。

である。又 […………] 内の DML は、点 r が木内の最右、最下の葉点の時、 T_p に出力される。

[[…………]] 内の DML は、点 r が葉点 (但し、最右の葉点ではない) の時、 T_p に出力される。

領域 B は、各点の目標項目値を 1 つ格納できる領域である。最右の葉点で、 B 内の全ての目標項目値の組が求ったとき、ある目標関数 t_l を B に適用 ($T \leftarrow t_l(B)$) して、目標集合 T の 1 つの組をつくり、出力テープに 1 組出力する。

表において、[[…………]] 内の DML は、[[…………]] 内の VRO 集合 X が、 T 内の複数の点によって表されている時に、出力される。これは、 T 内のある点 r_i と r_j が X を表しているとする。このとき、 r_i でアクセスされた時のテープ T_X のヘッド H_X の位置が、 r_j によるアクセスで H_X が動いてしまい、再び r_i で find next ……、演算を行うと、 r_j による H_X を 1 つ右に動かすことになる。以上のことは、VS 型 Y のテープ T_Y に対してもいえる (但し、 Y を複数の主辺が表しているときには、両端の点について前述したことがいえる)。従って、この時 r_i で find next ……を行う際には、まず、 H_X の位置を以前の位置に戻さねばならない。この為、 H_X を 1 つ動作させた時、その位置のセルの db キーを変数 δ_{r_i} に格納しておく。次に H_X (同時に H_Y) を動かす時は、まず H_X を、 δ_{r_i} 内の db キーを有しているセルに動かしてから行う様にする必要がある。

各点 r で、制限式 rf_r と上方結合式 uf_r がチェックされる。この為には、get X 命令でヘッド H_X 下のセルのデータ項目値を記憶領域 (レジスタ) β_r (DML プログラムでは X と名前がつく) に読み出し、 β_r 内のデータ項目の値について、条件式 rf_r を評価する (i.e. $rf_r(\beta_r)$)。又 r が下方結合式 df_r を持つ時は、 df_r が参照する r の項目集合 (ot_r) の値を、領域 v_r に出力しておく (move ot_r in X to v_r .)。 uf_r の他点の集合を $\{r_1, \dots, r_k\}$ とする。 r で VR 実現値 $x \in X$ が得られた (get X) 時、既に $v_{r_1}, \dots, v_{r_k}$ には、 uf_r を評価する為の各々点 $r_1, \dots, r_k$ のデータ項目値が格納されている。よって、 uf_r は、 β_r と $v_{r_1}, \dots, v_{r_k}$ 内のデータ項目値によって評価される。

各点が目標項目集合 tl_r をもつとき、出力テープのバッファ β_0 (DML プログラムでは T と名前づけられる) に、上方条件を満たす VR 実現値 $x \in X$ の項目 tl_r 値を書き出す。これは、move tl_r in X , to tl_r in T によってなされる。

例として、図 3.5 の CQG の巡航木 T について考える。この巡航木の例を、図 6.17 に示す。図 6.17 の木 T は、全ての出力点が T の最右の経路上にあるので、巡航木である。木 T を、次の順に訪ねる。即ち (er) (ER, e) (RE, r) (RE, er 1) (ER, e 1)。根及び主辺と点の対に

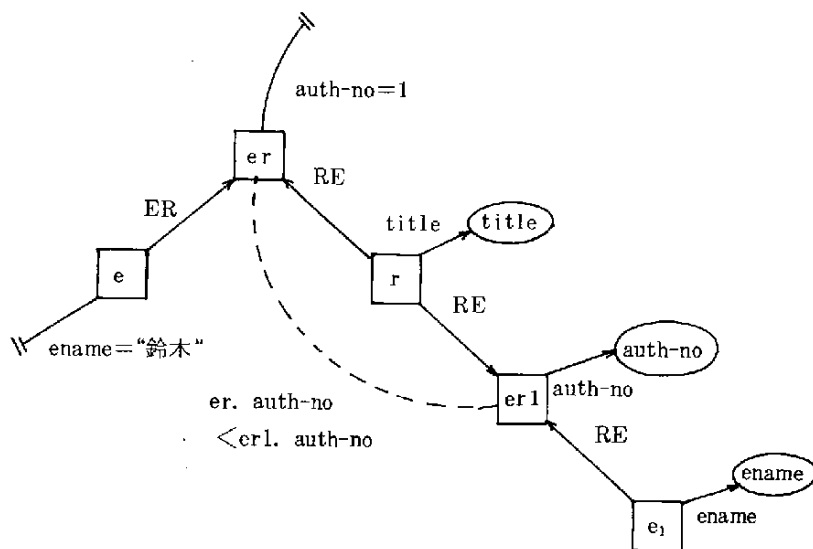


図 6.17 図 3.5 の巡航木例

対して、表 6.2 によって DML を出力すると図 6.18 の様な COBOL DML プログラムが生成できる。

```

open T. move o to mode. move o to TT.
  find first ERL within A. go to LERL.
KERL. find ERL ; db-key is DKERL.
MERL. find next ERL within A.
LERL. if SR=F go to EERL.
      get ERL. if not ( auth-no in ERL=1 ) go to MERL.
      accept DKER from ERL current.
      move auth-no in ERL to auth-no in BVERL.

      find owner within ER.
      if SR=F go to KERL.
      get EMP. if not ( ename in EMP="鈴木" ) go to KERL.
              go to SREP.

SREP.

      find owner within RE.
      if SR=F go to KERL.

```

get REP.

move title in REP to title in T.

SERL1. find first ERL within RE. go to LERL1.

KERL1. find ERL; db-key is DKERL1.

MERL1. find next ERL within RE.

LERL1. if SR=F go to KERL.

get ERL.

if not (auth-no in BVERL < auth-no in ERL) go to MERL1.

move auth-no in ERL to auth-no in T.

find owner within ER. if SR=F go to KERL1.

get EMP. move ename in EMP to ename in T.

if mode=1 and T=TT go to MERL1.

move 1 to mode. write T. move T to TT.

go to MERL1.

EERL. close T.

図 6.18 図 6.17 の DML プログラム

7. CQGの巡航木表現方法

本章では、任意のCODASYL問合せグラフ(CQG)Gを、ある目標を満たす巡航木Tで表すヒューリスティクスを示す。

7.1 目標と仮定

Gを任意のCODASYL問合せグラフ(CQG)とする。Gを表す巡航木をTとする。一般に、Gに対するTは、複数存在し、その数は、点と主辺の数の増加に対して、指数関数的に増加していくと考えられる。この為、Gに対して、ある目標を達成する様な1つのTを求めねばならない。我々は、この目標として、以下の2点を設ける。

〔目標〕

CQG Gを表す巡航木をTとする。Tを表すDMLプログラムをPとする。任意のGから、Pの実行応答時間(E_p)が最少となる様に、巡航木Tを求める。□

利用者にとって、何よりも良好な応答時間を得ることが望ましい。ここで、応答時間について以下の仮定を設ける。

〔仮定 7.1〕

DMLプログラムPの実行経過時間 E_p は次の様である。

$$E_p = a + b \cdot x + c \cdot y$$

ここで、aは、Pの初期化と終了化(データベースと出力ファイルのオープンとクローズ等)の為の経過時間であり、bとcはある定数である。xは、PがアクセスするVR実現値数である。yは、出力ファイルに書き出すレコード(目標集合の組)の数である。ここで、常に $y \leq x$ である。□

即ち、Pの実行したときの経過時間は、Pがアクセスする仮想データベースv内のVR実現値数と、Pが出力する目標集合のカーディナリティ(目標集合の組の数)に依存するとする。

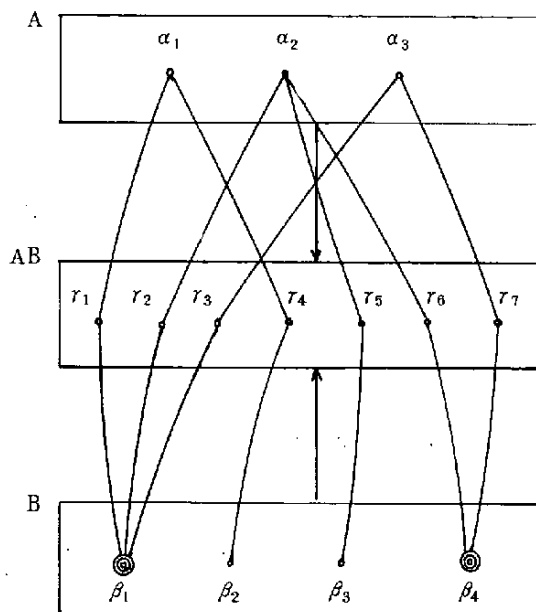
例として、図7.1を考えてみよう。図のa)はVR実現値の関連を示している。図のb)は、a)に対するCQGであるとする。又、VRO集合B内で、 rf_b を満足するVR実現値は、 β_1, β_4 であるとする。このとき、Aからアクセスしていくと、条件を満たすA、AB、B内のVR実現値の組($\alpha \in A, r \in AB, \beta \in B$)としては、次のものがある。即ち

$(\alpha_1, \gamma_1, \beta_1)$
 $(\alpha_2, \gamma_2, \beta_1)$
 $(\alpha_3, \gamma_3, \beta_1)$
 $(\alpha_2, \gamma_6, \beta_4)$
 $(\alpha_2, \gamma_7, \beta_4)$

の5組がある。よって、目標集合は $(\beta_1, \beta_1, \beta_1, \beta_4, \beta_4)$ の5組から成る。一方、Bからアクセスすると、 β_1, β_4 の2組だけが目標集合として生成される。

この様に、出力される目標集合の組数(カーディナリティ)は、巡航木の形によって異ってくる。この例からわかる様に、一般に、木内の上位に、目標点を置く方が、目標集合のカーディナリティを減少出来る。

a) 実現値



b) C Q G

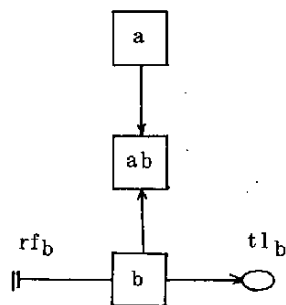


図 7.1 例

解を得る為にアクセスされるVR実現値数も、前者の場合は、 α_1 に対して、ABとBが各々2、 α_2 に対しては3、 α_3 に対しては2となり、計 $(1+2+2)+(1+3+3)+(1+2+2)=5+7+5=17$ 個のVR実現値がアクセスされる。一方、後者では、 β_1 と β_4 に対して、ABとB内に1つずつ、 β_2 と β_3 では rf_b を満たしたいので β_2 と β_3 だけがアクセスされる。よって、 $2(1+1+1)+2=6+2=8$ 個のVR実現値がアクセスされることになる。この様に、巡航木の形によってアクセスされるVR実現値数も異なることがわかる。

7.2 アクセスコストと出力コスト

ある巡航木Tが与えられたとき、そのコストを定義する。

〔定義 7.1〕

コストの次元は、VR実現値又は目標集合の組の数である。□

〔定義 7.2〕

巡航木Tのアクセスコスト $acost(T)$ とは、Tに対応した巡航的DMLプログラムPがアクセスする仮想データベースv内の仮想レコード(VR)実現値の数である。Tの出力コスト $ocost(T)$ とは、Tに対応したDMLプログラムPが、出力ファイルに出力する組(レコード)数、即ち、目標集合の組数である。□

〔定義 7.3〕

任意のCQG Gに対して、Gを正しく表す巡航木Tの集合を、Gの巡航木集合 T_G とする。□

CQG Gに対する巡航木集合 T_G 内の巡航木の数は、GのO(変数)点の数と、辺の数の増加に対して、指数関数的に増加してしまう。Gの T_G の中から、目標を満たすある巡航木(応答時間をなるべく小さくし、かつ目標集合のカーディナリティをなるべく小さくする巡航木)を見つける必要がある。以上より、CQG Gから、ある巡航木 $T \in T_G$ を求める問題は、次の様になる。

〔問題〕

入力 CODASYL問合せグラフ $G=(N, E)$

出力 巡航木 $T=(M, B)$

問題 Gから、次の条件を満たし、かつGを正しく表す巡航木 $T \in T_G$ を見つける。

$$\begin{aligned} & \min_{\forall T \in T_G} (E_p = a + b \cdot acost(T) + c \cdot ocost(T)) \\ & = \min_{\forall T \in T_G} (b \cdot acost(T) + c \cdot ocost(T)) \quad \square \end{aligned}$$

即ち、任意の CQG G に対して、アクセスされる VR 実現値の数 ($acost(T)$) 及び目標集合のカーディナリティ ($ocost(T)$) が最少な巡航木 T を、 G の巡航木集合 T_G から見つける必要がある。

7.3 コストの見積り

次に、巡航 T が与えられたとき、アクセスコスト $acost(T)$ と、出力コスト $ocost(T)$ の見積りについて考える。

A 基本定義

[定義 7.4]

巡航 T に対して、DMLG アルゴリズムによって T を表す DML プログラムを P とする。 P がアクセスする VR 実現値の期待数を、期待アクセスコスト $eacost(T)$ 、又、 P が出力するレコードの期待数を、期待出力コスト $ecost(T)$ とする。□

[定義 7.5]

巡航木 T 内の各点 x に対して以下を定義する。

$x \triangleq$ VR 型 X を表す点

$\Omega_x \triangleq \Omega_X \triangleq X$ の項目集合

$C_x \triangleq$ VRO 集合 X のカーディナリティ
 $\triangleq \text{card}(X)$

$\pi_x \triangleq \pi_X = \Omega_X - \{\odot X\}$

$\triangleq X$ のデータ項目の集合

各項目 $\tau \in \Omega_x$ に対して、

$\text{card}(\tau) \triangleq X$ 内で τ の一意な値の数 $\triangleq \text{card}(X[\tau])$
 $\triangleq \tau$ の定義域

□

[仮定 7.2]

平均アクセスコストと平均出力コストを考えるうえで、以下の仮定を設ける。

- (i) 仮想データベース v 内の各 VRO 集合を X とする。 X 内で、各項目 $\tau \in \Omega_X$ の値の分散は均一である。
- (ii) v 内で各項目の値の分散は、互いに独立である。

(III) v 内の各 VSO 集合を Y とし、その V S 型を $\underline{Y} = (\underline{X}_1, \underline{X}_2)$ とする。各親 VR 実現値 $x_1 \in X_1$ の子 VR 実現値集合を $m_Y(x_1)$ とする。 $m_Y(x_1) (\subseteq X_2)$ 内の $\underline{X}_2$ の各項目 $t \in \mathcal{Q}_{X_2}$ の値の分散は均一であり、 X_2 内の分散と同一である。

(IV) 巡航木 T 内の各結合辺 $j f_{xy}$ を考える。

$j f_{xy}$ は、次の形式の等結合述語から成るとする。

$$j f_{xy} ::= j p_{xy} \mid j p_{xy} \wedge j f_{xy}$$

$$j p_{xy} ::= x.t = y.u$$

$$\text{ここで、 } t \in \mathcal{Q}_X, u \in \mathcal{Q}_Y$$

$$\text{dom}(t) = \text{dom}(v)$$

(V) T 内の各点 x の制限式 $r f_x$ は、次の様に積正規化されているとする。

$$r f_x ::= r f_{xt} \mid r f_{xt} \wedge r f_x$$

$$r f_{xt} ::= r p_{xt} \mid r p_{xt} \vee r f_{xt}$$

$$r p_{xt} ::= x.t \theta v$$

ここで、 $\theta \in \{<, \leq, =, \geq, >, \neq\}$ 、 $t \in \pi_x (= \mathcal{Q}_x - \odot X)$ 、 v は定数。

即ち、

$$r f_x = (x.t_1 \theta_{11} v_{11} \vee \cdots \vee x.t_1 \theta_{1\ell_1} v_{1\ell_1}) \wedge \cdots \wedge \\ (x.t_p \theta_{p1} v_{p1} \vee \cdots \vee x.t_p \theta_{p\ell_p} v_{p\ell_p})$$

ここで

$$\theta_{ij} \in \{<, \leq, =, \geq, >, \neq\}。 \quad \square$$

[定義 7.6]

仮想レコード (VR) 型 $\underline{X} = (\mathcal{Q}_X)$ の VRO 集合 X に対して、以下を定義する。

各項目 $\tau \in \mathcal{Q}_X$ に対して、

$$\begin{aligned} j t_\tau &\triangleq [X \text{ 内の } \tau \text{ の一意な値の数}] / [\tau \text{ の可能な値の数}] \\ &\triangleq \text{card}(\tau) / \text{card}(\text{dom}(\tau)) \quad [\text{HEVNA 78}] \\ &\triangleq \text{項目 } \tau \text{ の結合選択度。} \end{aligned}$$

$j t_\tau$ は、定義域 $\text{dom}(\tau)$ 内の個々の値が、VRO 集合 X 内にもある確率である。

各項目 $\tau \in Q_X$ に対して、

$$\begin{aligned} st_{\tau} &\triangleq jt_{\tau} \cdot \text{card}(\tau) / C_X \\ &\triangleq \text{項目 } \tau \text{ の制限選択度。} \end{aligned}$$

st_{τ} は、VRO集合X内の各VR実現値の項目 τ の値が、定義域 $\text{dom}(\tau)$ 内のある値である確率である。 $\square$

項目 τ の選択度 st_{τ} を用いると、Xを表す点 x の制限式 rf_x を満足するX内のVR実現値数の期待値を求められる。例えば、 rf_x は、 $x.t=v$ であったとすると、この rf_x を満足するVR実現値の期待数 O_x は、次の様になる。

$$O_x = C_x \cdot st_{\tau}$$

[定義 7.7]

点 x が制限式 rf_x を有しているとする。この時 rf_x の選択度 st_x を以下を定める。

$$st_x \triangleq \text{VRO集合X内の各VR実現値が制限式 } rf_x \text{ を満足する確率。} \quad \square$$

種々の形式の制限式 rf_x に対する選択度 st_x は、各項目値の分散が均一でかつ互いに独立であることから、表 7.1 の様に求められる。

例えば、0変数 e を表すVR型 EMP ($eno, ename, \dots, sal, \dots$) を考える。ここで

$$\begin{aligned} C_e &= 1000 \\ st_{eno} &= 0.0009 & st_{sal} &= 0.02 \\ st_{ename} &= 0.0015 \end{aligned}$$

であるとする。この時、制限式 $rf_e(e.eno=10)$ の選択度 $st_e = st_{eno} = 0.0009$ となり、 rf_e を満たすVR実現数の期待値は、 $C_e \cdot st_e = 1000 \times 0.0009 = 0.9$ となる。又、 e についての制限式 $rf_e = (e.eno=10 \vee e.eno=8 \vee e.eno=500) \wedge e.sal \geq 100000$ の選択度 st_e は、表 7.1 より次の様になる。

$$\begin{aligned} st_e &= (st_{eno} + st_{eno} + st_{eno}) \cdot (1 + st_{sal}) / 2 \\ &= 3 \cdot st_{eno} \cdot (1 + st_{sal}) / 2 \\ &= 3 \cdot 0.0009 \cdot (1 + 0.02) / 2 \\ &= 3 \cdot 0.0009 \cdot 0.51 \\ &= 0.00138 \end{aligned}$$

よって、 rf_x を満たすVR実現値数の期待値は、 $1000 \times 0.00138 = 1.38$ (人)となる。

表 7.1 制限式 rf_x と選択度 st_x

rf_x	st_x
(1) $x, \tau = v$	st_τ
(2) $x, \tau \neq v$	$1 - st_\tau$
(3) $x, \tau_1 = v_1 \vee \cdots \vee x, \tau_n = v_n$	$n \cdot st_\tau$
(4) $rp_{x\tau_1} \wedge \cdots \wedge rp_{x\tau_l}$	$st_{x\tau_1} \cdot \cdots \cdot st_{x\tau_l}$ ここで、 $st_{x\tau_i}$ は、 $rp_{x\tau_i}$ の選択度
(5) $x, \tau \geq v$	$(1 - st_\tau) / 2$
(6) $x, \tau \leq v$	$(1 + st_\tau) / 2 = (1 - st_\tau) / 2 + st_\tau$

〔定義 7.8〕

各VS型 $\underline{Y} = (\underline{X}_1, \underline{X}_2)$ のVSO集合 Y に対して、結合度 ct_Y 及び逆結合度 it_Y が次の様に定義される。

$$\begin{aligned}
 ct_Y &\triangleq \text{各VR実現値 } x_1 \in X_1 \text{ が持つ子VR実現値 } x_2 \in m_Y(x_1) \text{ の数の期待値 } (\geq 0) \\
 &\triangleq \left\{ \sum_{\forall x_1 \in X_1} \text{card}(m_Y(x_1)) \right\} / \text{card}(X_1) \\
 it_Y &\triangleq \text{各VR実現値 } x_2 \in X_2 \text{ が、ある親VR実現値 } x_1 (x_2 \in m_Y(x_1)) \text{ を持つ確率} \\
 &\triangleq [\text{card}(\{x_1 \mid m_Y(x_1) \neq \emptyset\})] / \text{card}(X_1)。
 \end{aligned}$$

ct_Y と it_Y には、次の関係が成り立つ。

$$ct_Y \triangleq it_Y \cdot \text{card}(X_2) / \text{card}(X_1) \quad \square$$

即ち、各 $x_1 \in X_1$ は、平均 ct_Y 個の子VR実現値を X_2 内に持つ。 X_2 の中で、親を持つVR実現値の期待数は、 $\text{card}(X_2) \cdot it_Y$ となる。 $mc(\underline{Y}) = M/A$ の時、全ての子は必ず親をもつので、 $it_Y = 1$ である。この時は $ct_Y = \text{card}(X_2) / \text{card}(X_1)$ となる。

〔定義 7.9〕

巡航木 T の各枝 $S(r_i, r_j)$ に対して、主辺 S の結合度 bt_S を以下の様に定義する。

$$bt_S = \begin{cases} ct_Y & \text{if } r_j \text{ はM型} \\ & (r_j \text{ は、} S \text{ の息子点}) \\ it_Y & \text{otherwise } (r_j \text{ がO型}) \end{cases} \quad \square$$

図 6.17 の巡航木で、O 点 er と e 間の主辺 ER に対して、 $bt_{ER} = it_{ER}$ である。又、O 点 r と er 間の主辺 RE に対して、 $bt_{RE} = ct_{RE}$ である。

次に、巡航木 T 内の結合辺について考える。 T が巡航木であることから、点 r_i と r_j とは、同一の上下路内に存在する。ここで、点 r_i が根点により近いとする〔図 7.2〕。 r_i と r_j 間の結合辺

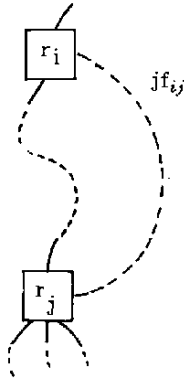


図 7.2 結合辺 jf_{ij}

jf_{ij} は、 T の上方条件を満足するある VR 実現値 $x_i \in X_i$ (i. e. $uecond_i(x_i)$) に対して、 X_j 内の VR 実現値 x_j が、結合式 $jf_{ij}(x_i, x_j)$ を満足するかどうかチェックされる。よって、結合式 jf_{ij} は、点 r_j においては制限式の役割をもつ。ここで、 $jf_{ij} :: r_i, t_i = r_j, t_j$ について考える。 $D = \text{dom}(t_i) = \text{dom}(t_j)$ とする。この時、結合式 $r_i, t_i = r_j, t_j$ を、定義域 D 内の各値が満たす確率は、 v が VRO 集合 X_i にも X_j 含まれる確率、即ち、 $jt_{t_i} \cdot jt_{t_j}$ となる。よって

D 内の結合式を満たす値の数 (N) は次の様になる。

$$\begin{aligned}
 N &= \text{card}(D) \cdot jt_{t_i} \cdot jt_{t_j} \\
 &= \text{card}(D) \cdot \frac{\text{card}(t_i)}{\text{card}(D)} \cdot \frac{\text{card}(t_j)}{\text{card}(D)} \\
 &= \text{card}(t_i) \cdot \frac{\text{card}(t_j)}{\text{card}(D)} \\
 &= \text{card}(t_i) \cdot jt_{t_j} \\
 &= \text{card}(t_j) \cdot jt_{t_i} \text{。}
 \end{aligned}$$

結合 $r_i, t_i = r_j, t_j$ 後、VRO 集項 X_j 内でこの結合式を満たす VR 実現値の期待数 E_{r_j} は、

$$E_{r_j} = \text{card}(t_j) \cdot jt_{t_i} \cdot jt_{t_j} \cdot \text{card}(D)$$

となる。一方、 X_i 内の t_i の個々の値に対して X_j 内の個々の VR 実現値が、この結合式を満たす確率 P_{t_j} は、次の様である。

$$\begin{aligned}
 P_{t_j} &= \text{card}(t_j) \cdot jt_{t_i} \cdot jt_{t_j} / \text{card}(X_j) \\
 &= \frac{\text{card}(t_j)}{\text{card}(X_j)} \cdot jt_{t_j} \cdot jt_{t_i}
 \end{aligned}$$

$$= st_{t_j} \cdot jt_{t_i} \circ$$

従って、 $r_i, t_i = r_j, t_j$ の結合によって、 X_j の項目 t_j の(制限)選択度 st_{t_j} と、結合選択度 jt_{t_j} は、次の様になる。

$$st_{t_j} \leftarrow st_{t_j} \cdot jt_{t_i} ;$$

$$jt_{t_j} \leftarrow jt_{t_j} \cdot jt_{t_i} ;$$

次に、結合式 $jf_{ij} = \bigwedge_{k=1}^l (r_i, t_{ik} = r_j, t_{jk})$ について考える。 X_i の $t_{i1}, \dots, t_{il}$ の個々の値に対して X_j 内の各 VR 実現値が、 jf_{ij} を満たす確率 P_j は、個々の結合述語 $r_i, t_{ik} = r_j, t_{jk}$ に対する確率 P_{jk} の積となる。

$$P_j = P_{j1} \cdots P_{jl} = \prod_{k=1}^l P_{jk} \circ$$

結合式 jf_{ij} によって、 X_j 内の各項目 τ の制限度と結合選択度とは、次の様になる。

各 $t_{jk} (\in Q_{X_j})$ に対して、($k = 1, \dots, l$)

$$st_{t_{jk}} \leftarrow st_{t_{jk}} \cdot jt_{t_{ik}} ;$$

$$jt_{t_{jk}} \leftarrow jt_{t_{jk}} \cdot jt_{t_{ik}} ;$$

各 $t \in Q_{X_j} - \{t_{j1}, \dots, t_{jl}\}$ に対して、 st_τ, jt_τ は変わらない。

〔定義 7.10〕

結合式 $jf_{ij} = \bigwedge_{k=1}^l (r_i, t_{ik} = r_j, t_{jk})$ を表す巡航木 T 内の結合辺 jf_{ij} を考える。結合辺 jf_{ij} の根に近い(上位)の点を r_i とし、葉に近い(下位)の点を r_j とする。

このとき、結合辺 jf_{ij} による下位点 r_j の結合選択度 jt_{ij} を、次の様に定義する。

$$jt_{ij} \triangleq \prod_{k=1}^l st_{t_{jk}} \cdot jt_{t_{jk}}$$

$\triangleq$ jf_{ij} による点 r_j の結合選択度。

$\triangleq$ X_i の各 VR 実現値 x_i に対して、 X_j 内の各 VR 実現値 x_j が、結合条件 jf_{ij} (x_i, x_j) を満たす確率

□

例として、図 7.3 を考える。

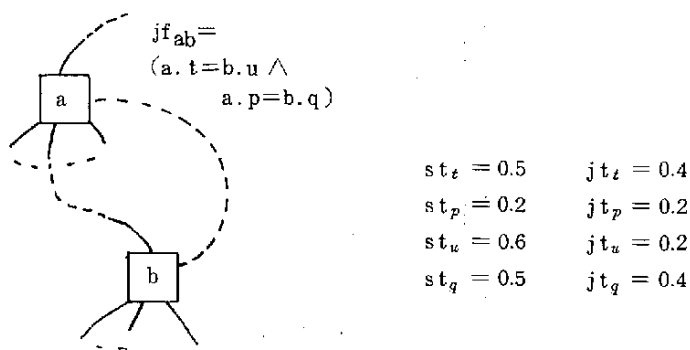


図 7.3 結合式

このとき、 jf_{ab} の b の a による結合選択度 jt_{ab} は次の様になる。

$$\begin{aligned}
 jt_{ab} &= (st_u \cdot jt_t) \cdot (st_q \cdot jt_p) \\
 &= (0.6 \cdot 0.4) \cdot (0.5 \cdot 0.2) \\
 &= 0.024
 \end{aligned}$$

結合後の b の項目 u と q の選択度、結合選択度は次の様になる。

$$\begin{aligned}
 st_u &= 0.6 \cdot jt_t = 0.6 \cdot 0.4 = 0.24 \\
 jt_u &= 0.2 \cdot jt_t = 0.2 \cdot 0.4 = 0.08 \\
 st_q &= 0.5 \cdot jt_p = 0.5 \cdot 0.2 = 0.1 \\
 jt_q &= 0.4 \cdot jt_p = 0.4 \cdot 0.2 = 0.08
 \end{aligned}$$

巡航木 T の各点 r は、 r から根までの径路上の k (≥ 0) 個の点 $r_1, \dots, r_k$ と各々結合辺 $jf_1, \dots, jf_k$ を持ち、これらの辺の集合を uf_r とする〔図 7.3〕。結合辺 uf_r は、 r の上方の結合式 uf_r を表す。ここで、次の仮定を設ける。

〔仮定 7.3〕

uf_r 内の r の (結合) 項目集合 $A_r = \{t_1, \dots, t_k\} \subseteq \mathcal{Q}_r$ とする。このとき、各結合項目 $\tau \in A_r$ は、ただ 1 つの点 r_i との結合述語 r_i 、 $\tau_i = r_i, \tau$ にのみ含まれる。

このとき、 r の各結合項目 $\tau \in A_r$ に対して、

$$jnode(\tau) = r_i$$

$$jtem(\tau) = \tau_i$$

とする。□

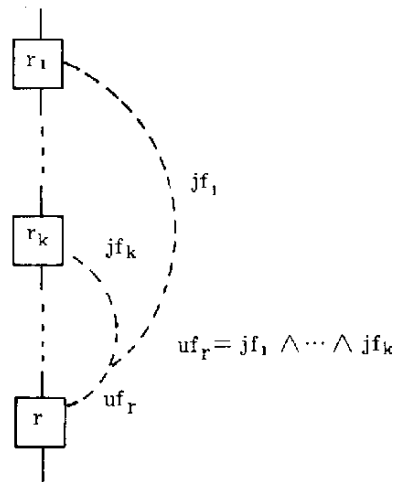


図 7.4 uf_r の結合選択度

〔定義 7.11〕

点 r が、 r から根までの径路のある点 $r_1, \dots, r_k$ 間に各々結合辺 $jf_1, \dots, jf_k$ が存在するとする〔図 7.4〕。即ち、 $uf_r = jf_1 \wedge \dots \wedge jf_k$ である。各結合辺 jf_i は、点 r に対する結合選択度 jt_i を有しているとする。この時

$$jt_r \triangleq \prod_{i=1}^k jt_i$$

$\triangleq uf_r$ の結合選択度

uf_r 内の結合項目集合 $A_r = \{t_1, \dots, t_k\} \subseteq \mathcal{Q}_r$ とする。又、各 $t_i \in A_r$ に対して、 $jnode(t_i) = b_i$ 、 $jtem(t_i) = v_i \in \mathcal{Q}_{r_i}$ ($i = 1, \dots, k$) とする。ここで、 $\bigcup_{i=1}^k \{jnode(t_i)\} = \{r_1, \dots, r_k\}$ である。このとき、 r の VRO 集合 X 内の VR 実現値が、点 $r_1, \dots, r_k$ 内の条件を満たす 1 組の VR 実現値 $(x_1, \dots, x_k)$ に対して、条件 $uf_r(x_1, \dots, x_k, r)$ を満たす確率 ut_r は、次の様になる。

$$ut_r \triangleq \prod_{\forall t_i \in A_r} st_{t_i} \cdot jt_{u_i} \quad (u_i = jtem(t_i))$$

この結果、 r の各項目 r の選択度 st_r 、結合選択度 jt_r は、次の様になる。

各 $t_i \in A_r$ に対して

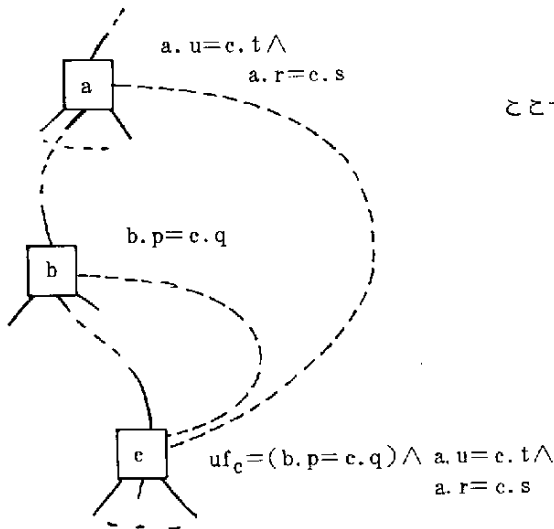
$$st_{t_i} \leftarrow st_{t_i} \cdot jt_{u_i} \quad (u_i = jtem(t_i))$$

$$jt_{t_i} \leftarrow jt_{t_i} \cdot jt_{u_i}$$

各 $\tau \in \mathcal{Q}_r - \{A_r\}$ (結合項目でない) は、

$st_\tau \leftarrow st_\tau ;$

$st_\tau \leftarrow jt_\tau ;$ である。 $\square$



ここで、

$st_u = 0.5$	$jt_u = 0.3$
$st_t = 0.2$	$jt_t = 0.2$
$st_r = 0.8$	$jt_r = 0.4$
$st_s = 0.5$	$jt_s = 0.4$
$st_p = 0.6$	$jt_p = 0.5$
$st_q = 0.8$	$jt_q = 0.4$

図 7.5 選択度の例

例として、図 7.5 を考える。点 c の上方の結合式 $jf_c = (b.p=c.q \wedge a.u=c.t \wedge a.r=c.s)$ の結合選択度 ut_c は、次の様になる。

$$\begin{aligned}
 ut_c &= (st_q \cdot jt_p) \cdot (st_t \cdot jt_u) \cdot (st_r \cdot jt_s) \\
 &= (0.8 \cdot 0.5) \cdot (0.2 \cdot 0.3) \cdot (0.8 \cdot 0.4) \\
 &= 0.4 \quad \cdot \quad 0.06 \quad \cdot \quad 0.32 \\
 &= 0.0077
 \end{aligned}$$

この結合後、 c の各結合属性の選択度、結合選択度は、次の様になる。

$$\begin{aligned}
 st_q &= 0.8 \quad \cdot \quad jt_p = 0.4 \\
 st_t &= 0.2 \quad \cdot \quad jt_u = 0.06 \\
 st_r &= 0.8 \quad \cdot \quad jt_s = 0.32 \\
 jt_q &= 0.4 \quad \cdot \quad jt_p = 0.2 \\
 jt_t &= 0.2 \quad \cdot \quad jt_u = 0.06 \\
 jt_r &= 0.4 \quad \cdot \quad jt_s = 0.16
 \end{aligned}$$

7.4 巡航木の期待アクセスコスト

次に、巡航木 T が与えられたときの、 T の期待アクセスコスト $eacost(T)$ を求める。期待アクセスコストは、巡航木 T 内の各点と辺についての次のパラメータをもとにして計算される。

[$eacost(T)$ のパラメータ]

(i) 巡航木 T の各点に対して、

$C_r \triangleq r$ の表す VRO 集合 X のカーディナリティ ($card(X)$)

$\mathcal{Q}_r \triangleq X$ の項目集合

$tl_r \triangleq r$ の有する目標項目の集合 ($\subseteq \mathcal{Q}_r$) (なければ、 $\emptyset$ である)

$rf_r \triangleq r$ の有する制限式 (なければ、 $\emptyset$ とする)

$tr_r \triangleq rf_r$ が参照する X のデータ項目集合 ($rf_r = \emptyset$ ならば、 $= \emptyset$)

$uf_r \triangleq$ 点 r の上方の結合式 (なければ $\emptyset$)

$tu_r \triangleq uf_r$ が参照する X のデータ項目集合 (なければ、 $\emptyset$)

各項目 $\tau \in \mathcal{Q}_r$ に対して

$jt_\tau \triangleq$ 項目 τ の結合選択度

$st_\tau \triangleq$ 項目 τ の選択度。

各点 r について

$\alpha_r \triangleq tr_r$ 内の項目の選択度を用いて求めた rf_r の選択度

(ii) T の各主辺 $S(r_i, r_j)$ に対して

$bt_S \triangleq$ 枝 S の結合度 □

図 7.6 の様な巡航木 T を考える。 r_0 は T の根であり、 r_i は T 内のある点とする。 r_0 と r_i 間には、径路 $r_0, S_1, r_1, \dots, S_{i-1}, r_{i-1}, S_i, r_i$ が存在するとする。点 r_i は、 n_i (≥ 0) 個の息子 $r_{i1}, \dots, r_{in_i}$ を持ち、各々は部分木 $T_{i1}, \dots, T_{in_i}$ の根となるとする。この時、部分木 T に対して、根 r_i の条件 (rf_i と uf_i) を満たすある 1 つの VR 実現値からアクセスされる VR 実現値数の期待数 $ec(T_i)$ を求める。

[定義 7.1.2]

$ec(T_i) \triangleq T_i$ の根 r_i の表す VRO 集合 X_i の各 VR 実現値 $x_i (\in X_i)$ から、部分木 T_i 内でアクセスされる VR 実現値の期待数

$$\triangleq \begin{cases} 1 + \alpha_i \cdot ut_i \cdot [bt_{S_{i1}} \cdot ec(T_{i1}) + \\ \quad \varphi_{T_{i1}} \cdot [bt_{S_{i2}} \cdot ec(T_{i2}) + \\ \quad \dots \\ \quad \varphi_{T_{in_i-1}} [bt_{S_{in_i}} \cdot ec(T_{in_i})] \dots]] \text{ if } n_i > 0 \\ 1 \quad \text{otherwise} \end{cases}$$

$\varphi_{T_i} \triangleq$ 部分木 T_i の選択度

$$\triangleq \begin{cases} \alpha_i \cdot ut_i \cdot \eta_{T_{i1}} \cdot \dots \cdot \eta_{T_{in_i}} & \text{if } n_i > 0 \\ \alpha_i \cdot ut_i & \text{otherwise } (n_i = 0) \end{cases}$$

$\triangleq$ X_i の各VR実現値 x_i が、 x_i の下方の条件 $dcond_i(x_i) = \text{true}$ である確率

$\zeta_{T_i} \triangleq$ 各 $x_{i-1} \in X_{i-1}$ に対して、 $B_{S_i}(x_{i-1}, x_i)$ を満たす $x_i \in X_i$ であつ、 x_i の下方条件 (T_i) を満たす x_i の期待数

$$\triangleq bt_i \cdot \varphi_{T_i}$$

$$\eta_{T_i} \triangleq \begin{cases} 1 & \text{if } \zeta_{T_i} \geq 1 \\ \zeta_{T_i} & \text{otherwise } (\zeta_{T_i} < 1) \end{cases}$$

$\triangleq$ 各 $x_{i-1} \in X_{i-1}$ に対して、下方条件 $dcond_i(x_i)$ を満たし、かつ $B_{S_i}(x_{i-1}, x_i)$ ($x_{i-1} \in X_{i-1}$) である $x_i \in X_i$ が少くとも 1 つ存在する確率

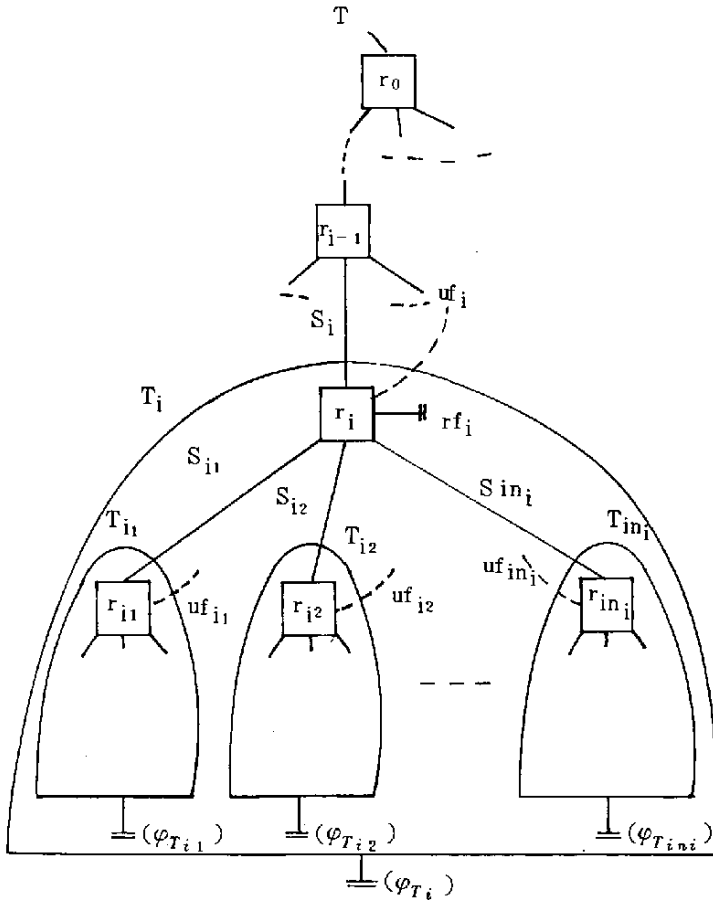


図 7.6 巡航木

次に、巡航木 T の期待アクセスコスト $eacost(T)$ を求める。 T の根 r_o の表す VRO 集合 X_o に対して、各 $x_o \in X_o$ からアクセスされる T 内の VR 実現値の期待数は、今与えた $ec(T)$ である。よって $eacost(T)$ は、条件 rf_o を満たす $x_o \in X_o$ を全て見つける為にアクセスした X_o 内の全 VR 実現値の期待数 N_o と $ec(T)$ の積となる。 N_o は、 X_o のアクセス方法で異なる。エリア内での全面サーチの場合には、 $N_o = C_o$ である。一方、CALC 項目を用いた場合には、 $N_o = 1$ である。

[定義 7.13]

$$eacost(T) \triangleq \begin{cases} C_r \cdot C(T) & \text{if } X_o \text{ のエリアの全面サーチの場合} \\ C(T) & \text{if CALC によるアクセス} \end{cases}$$

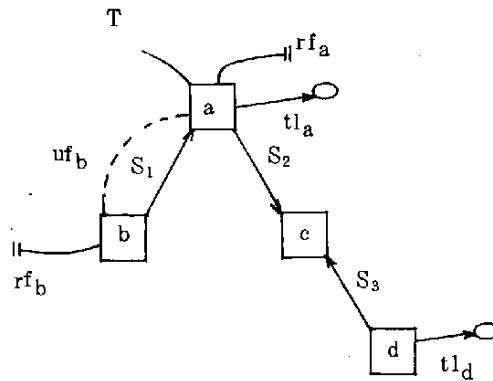


図 7.7 巡航木 T の例

例として、図 7.6 の点 a を根とする巡航木 T について考える。この T の期待アクセスコスト $eacost(T)$ は、次の様になる（ここで、点 a のアクセスパターンは、エリアの全面サーチとする）

$$\begin{aligned} eacost(T) &= C_a \cdot [1 + \\ &\quad st_a \cdot [its_1 + \\ &\quad (its_1 \cdot st_b \cdot ut_b) \cdot ct_{S_2} [1 + its_3]]] \\ &= C_a \cdot [1 + st_a \cdot its_1 \\ &\quad + st_a \cdot its_1 \cdot st_b \cdot ut_b \cdot ct_{S_2} \\ &\quad + st_a \cdot its_1 \cdot st_b \cdot ut_b \cdot ct_{S_2} \cdot its_3] \end{aligned}$$

7.5 巡航木の期待出力コスト

巡航木 T の期待出力コスト $eocost(T)$ について考える。巡航木 T では、定義より、全ての目標項目は、 T の幹内にある。 T の幹を r_o を根として、 $r_o, S_1, r_1, \dots, S_k, r_k$ とする〔図 7.8〕。各点 r_i は、図の様に幹外の部分木 $T_{i1}, \dots, T_{in_i}$ ($n_i \geq 0$) をもっている点 r_i の表す

VR0集合を X_i とする($i=0, 1, \dots, k$)。Tで rf_0 を満たす各VR実現値 $x_0 \in X_0$ に対して

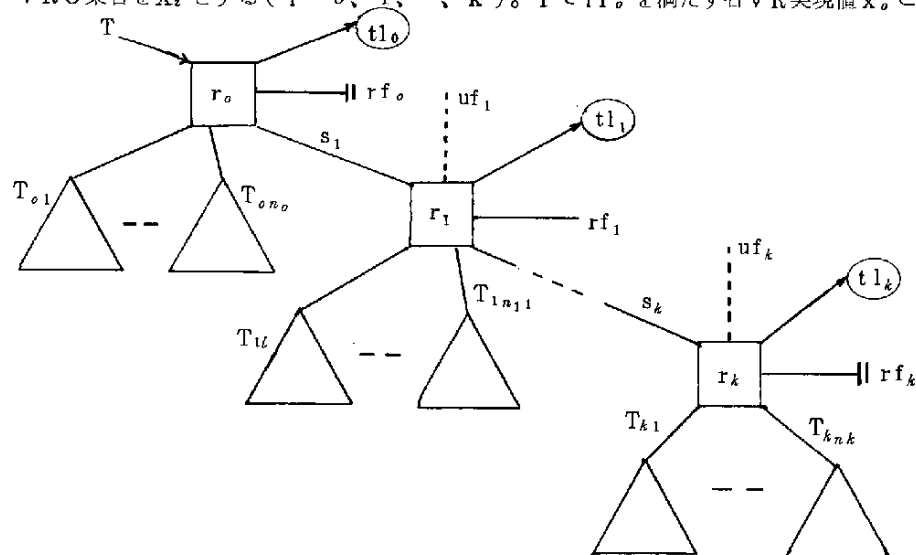


図 7.8 巡航木と幹

Tの条件を満たす $x_k \in X_k$ が求った時点で、1組の $x_0, x_1, \dots, x_k$ を、各目標項目集合で射影した値の組($tl_0(x_0), tl_1(x_1), \dots, tl_k(x_k)$)が、目標集合の1つの組として、出力テープに書かれる。よって、目標集合のカーディナリティは、 X_k 内の r の条件を満足するVR実現値数となる。ここで、 $S(T)$ を、1つのVR実現値 $x_0 \in X_0$ に対して、条件を満足する $x_k \in X_k$ の期待数とする。

〔定義 7.6〕

$$\begin{aligned}
 S(T) &\triangleq \alpha_0 \cdot bt_1 \cdot ut_1 \cdot \alpha_1 \cdot \varphi_1 \cdot \\
 &\quad bt_2 \cdot ut_2 \cdot \alpha_2 \cdot \varphi_2 \cdot \dots \cdot \\
 &\quad bt_k \cdot ut_k \cdot \alpha_k \cdot \varphi_k \\
 &\triangleq \alpha_0 \prod_{i=1}^k (bt_i \cdot ut_i \cdot \alpha_i \cdot \varphi_i) \\
 \varphi_i &\triangleq \begin{cases} \varphi_{T_{i1}} \cdot \varphi_{T_{i2}} \cdot \dots \cdot \varphi_{T_{in_i}} & \text{if } n_i > 0 \\ 1 & \text{otherwise } (n_i = 0) \end{cases} \\
 eocost(T) &= C_0 \cdot S(T) \quad \square
 \end{aligned}$$

7.6 巡航木生成アルゴリズム

CODASYL問合せグラフ(CQG)Gから、7.1の目標(即ち、 $\min_{T \in T_G} (b \cdot eocost(T))$)

$+c \cdot \text{eocost}(T))$ を満たす T を求めることが、次の問題となる。 G を表す全ての巡航木の集合 T_G について、目標を満たす T を見つけることは、 G の点と辺の増加に伴い、指数関数的に計算量が増大してしまうので不可能である。この為、ヒューリスティクスの導入が必要になる。 G に対する巡航木 T は、次の要求を満たすものでなければならない。

〔要求〕

- (i) G と T とは等価である(i. e. T は、 G を表す)。
- (ii) T の幹内に、全ての目標点は含まれねばならない。
- (iii) T 内の全ての結合辺に接する两点は、同一の上下路に含まれねばならない。
- (iv) $\text{eacost}(T)$ と $\text{eocost}(T)$ とをなるべく小さくせねばならない。 □

以下の我々の用いている巡航木生成ヒューリスティクス(DF(depth-first)アルゴリズム(DFA-AC))を示す。DFA-ACは、アクセスされるVR実現値数の減少を目指している。ここで、 CQG を $G=(N, E)$ とする。 G 内の全ての点 $x \in N$ と辺 $e \in E$ は、マーク($\text{mark}(x)$ と $\text{mark}(e)$)を持つ。はじめ、全ての点と辺のマークは0とする。

〔DFA-AC〕

- (1) 全ての0点 $x \in N$ の中で、アクセスされるVR実現値数 rcost を最少とする点を見つける
 $(\min_{x \in N} \text{rcost}(x))$ 。これを x とする。 x のマーク($\text{mark}(x) \leftarrow 1$)。 x を T の根とし、 G 内の x についての目標辺、制限辺を T に加える。 x に接する結合辺のマークを1とする。
- (2) x から、マークが0の全ての目標点 y までの径路 $P=(x, S_1, x_1, \dots, S_k, x_k, S_y, y)$ (単純径路とは限らない)を見つける。これらの中で、アクセスされるVR実現値数を最少とする P を見つける。即ち

$$\begin{aligned} & \min_{\substack{y \in N \text{ (} y \text{ は目標点 } \wedge \\ \text{mark}(y)=0 \text{)}}} \text{eacost}(P)。 \\ & \forall p \text{ (} p \text{ は } x \text{ と } y \text{ の径路) } \end{aligned}$$

- (3) 径路 p が見つかったとき、 p 内の全ての辺 S_i のマーク($\text{mark}(S_i) \leftarrow \text{mark}(S_i) + 1$)する。 $\text{mark}(S_i) \geq 2$ のとき、点 x_i と y のマーク($\text{mark}(x_i) \leftarrow \text{mark}(x_i) + 1$; $\text{mark}(y) \leftarrow \text{mark}(y) + 1$)する。径路 p を x_i が点 x の第1番目の息子となるように、かつ y への上下路として T に加える。

p 内の x 以外の点 z の $\text{mark}(z)=1$ の時、 G 内の x についての目標辺、制限辺と T 内に z に接する結合辺 e に対して、 $\text{mark}(e) \leftarrow \text{mark}(e) + 1$ とする。 $\text{mark}(e)=2$ のとき、この z と z' 間の接合辺を、 T 内で $\text{mark}(1)$ の z' と z 間の結合辺として加える。

p 内 x 以外の点 z のマークが 2 以上のとき、 T 内の z と、 $\text{mark}(z)=1$ なる T 内の z 間に $\odot z = \odot z$ なる結合辺を T に加える。 $x \leftarrow y$ として、(2)へ。

- (4) [マークが 0 の目標点の存在しないとき] x に接する全ての結合辺 e で、 $\text{mark}(e) \leq 1$ を求める。この様な e がある時、 e に接する点を y とする。点 x から y までの $\text{eacost}(p)$ を最少とする径路 p を求める。 p が求ったならば、(3)へ。
- (5) [x に接する結合辺がないか、又は全ての結合辺のマークが 2 以上であるとき] x に接するマークが 0 の y 間の主辺 S の中で、 x から y への主辺 S の結合度 bt_S が最少のものを見つける。見つかったならば、 $p = (x, S, y)$ として、(3)へ。
- (6) [x に接するマーク 0 の主辺のないとき] x が B 型でない親を持てば、 x の親を x として、(2)へ。
 x の親が B 型のとき、 x の親を x として、(6)へ。
- (7) [x が根のとき]
- 終了する。 □

[定理 7.1]

任意の CQG G から、 $DFA-AC$ によってつくられた CQG T は G と等価であり、かつ巡航木である。

[証明]

G と T 内の全ての目標辺、制限辺が 1 対 1 対応することは $DFA-AC$ より明らかである。 T 内の全ての主辺と結合辺の表す論理式から、結合述語 $x, \odot X = y, \odot X$ があるとき、 $x = y$ と簡約化すると G と同一になる。これは、 G 内のある点 x が、複数回訪ねられ、 T 内に、 $x_1, \dots, x_n$ ($n \geq 2$) の順に組みこまれたとき、必ず T 内には結合辺 $x_1, \odot X = x_i, \odot X$ ($i = 2, \dots, n$) を加えているからである。よって、 T と G とは等価になる。

T では、全ての目標点は、 T の最右の上下路内にある。又、全ての G 内の結合辺も、同一上下路内の点間にある。又、 G 内のある点 x にする T 内の点が $x_1, \dots, x_n$ ($n \geq 2$) の順に T 内に組み込まれた時、 $DFA-AC$ のサーチ方法より、必ず点 $x_2, \dots, x_n$ は、 x_1 を根とする部分木内にある。部分木内の x_i と根 x_1 とは同一上下路内にあるので、結合辺 $x_1, \odot X = x_i, \odot X$ は巡航木の条件を満たす。

以上より、本定理は成り立つ。 ■

[定義 7.7]

CQG G 内の全ての点 x に対して、 $\text{rcost}(x)$ は、 x の条件 rf_x を満たす全ての VR 実現値をみつける為にアクセスされる VR 実現値の期待数とする。 $\text{rcost}(x)$ は表 6.2 のパターン a)、b)、c) によって異なる。

- (i) a) の時、 VR O 集合 X を全面サーチするので、
- $$\text{rcost}(x) \triangleq C_x = \text{card}(X)$$

(ii) b)の時

$$\text{rcost}(x) \triangleq C_x \cdot st_x = 1$$

(iii) c)の時、ソートされているのでX内の半分のVR実現値をサーチするだけでよい。

$$\text{rcost}(x) \triangleq C_x / 2$$

同様に、 $\text{scost}(x)$ は、 rf_x を満たす全VR実現値数の期待値とすると。

$$\text{scost}(x) \triangleq C_x \cdot st_x$$

となる。 $\square$

一方、目標集合のカーディナリティを減少させる為のアルゴリズムDFA-OCを、次に示す。

(DFA-OC)

DFA-ACの(1)で、 $\text{rcost}(x)$ のかわりに $\text{S}_{\text{ost}}(x)$ の最少化を目指す。

(2)で、幹 p をみつけるとときには、 $\text{eocost}(p)$ を最少化する様にする。他は同じである。 $\square$

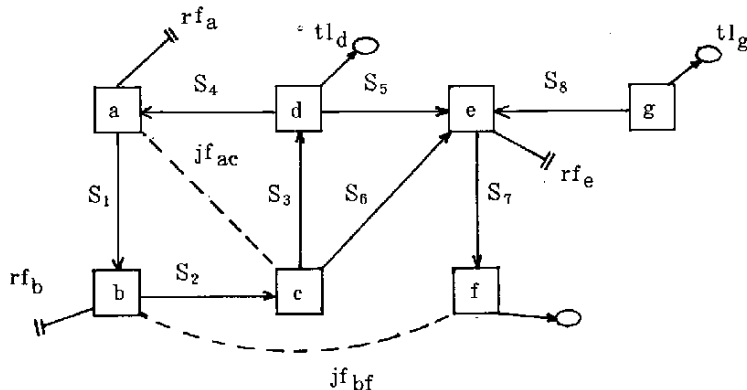


図 7.9 CQG の例

例として、図 7.9 の CQG G を考える。まず点 a の $\text{rcost}(a)$ が最少とする。 a から G 内の出力点集合 $O = \{d, f, g, \}$ を全て通る経路 p の中で、 $\text{eacost}(p)$ の最少のものをさがす。経路 $p = (a, S_4, d, S_5, e, S_7, f, S_8, g)$ が見つかったとする。この時、経路 p 内の点 a, d, e, f, g のマークは、次の様になる。

$$\text{mark}(a) = \text{mark}(d) = \text{mark}(f) = \text{mark}(g) = 1$$

$$\text{mark}(e) = 2$$

又、 p 内の主辺 S_4 、 S_5 、 S_7 、 S_8 のマークは、次の様になる。

$$\text{mark}(S_4) = \text{mark}(S_5) = \text{mark}(S_8) = 1$$

$$\text{mark}(S_7) = 2$$

この為、 p 内の S_7 の子息点 e は B となる〔図 7.10〕

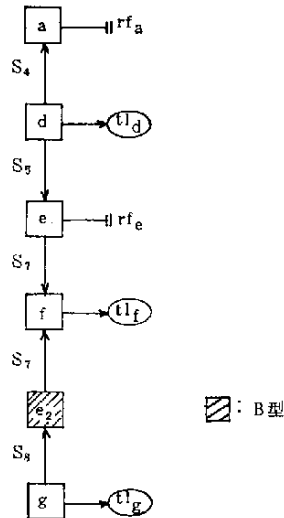


図 7.10 図 7.7 の幹

次に、 g は葉なので、 g_2 の父 e にもどる。この e_2 は B 型なので、 e_2 の親 f にもどる。 f は結合辺 jf_{bf} を持つ。この為、 f から b までの径路 p_{fb} で $\text{eacost}(p_{fb})$ の最少のものをもとめる。ここで、 $p_{fb} = (f, S_7, e, S_8, c, S_2, b)$ が求ったとする。この p_{fb} を、図 7.10 の f の第 1 子として加えると図 7.11 となる。

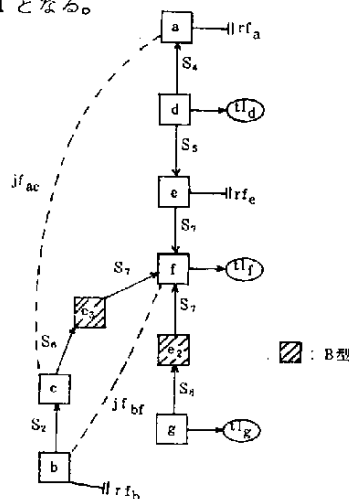


図 7.11

b は、他に結合辺がないので、b に接するマークが O の主辺を見つける。S₁ がわかるので、木に加える。点 a は概に木の根主であるので、根の a とこの a (a₂) 間に主辺 a. @A = a₂. @A を設ける。a₂ には rf_a をつけない。CQG の a は、もうマークの O の主辺と接しないので、木の a の親 b、c ともどる。c は、マーク O の主辺 S₃ をもつので、木の c の第 1 子息として加える。この d (= d₂) も概に木内になるので c と同じく、木内の d と d₂ 間に主辺 d. @D = d₂. @D を設ける。最終的に、図 7.12 に示す巡航木が得られる。図からわかる様に、全ての目標点は木の幹 (a、S₄、d、S₅、e、S₇、e₂、S₇、g) 内にあり、全ての結合辺は同一上下路内の点に接している。

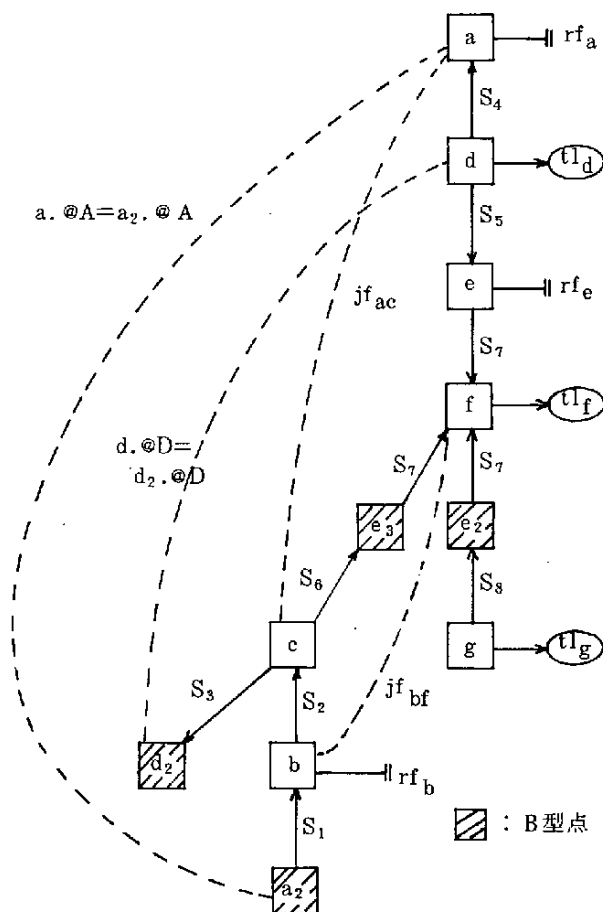


図 7.12 図 7.9 の DFA による巡航木

8. ローカルデータベースプロセッサ (LDP-V3)

本章では、統合型設計方法による異種分散型データベースシステムJDDBS〔TAKIM78～82〕のローカルデータベースプロセッサ(LDP-V3)について述べる。LDPは、異種DBSとしてのCODASYL DBS〔CODAS73〕(e. g. AIM(FACOM)、ADBS(NEC))上の関係インタフェースシステム〔TAKIM79、80、82b、83〕である〔図8.1〕。LDPは、CODASYLスキーマ(LIS)から等価な関係スキーマ(LCS)を生成し、かつ、関係スキーマ上の関係操作演算(検索と更新)を、CODASYL DBS上で実行可能なCOBOL DML〔OLLET73〕に変換し、DBSで実行させ、結果を関係としてRWS(関係作業領域)に格納するシステムである〔図8.2〕。LDPは、更新と検索演算を処理出来るが、ここでは検索についてのみ述べる。

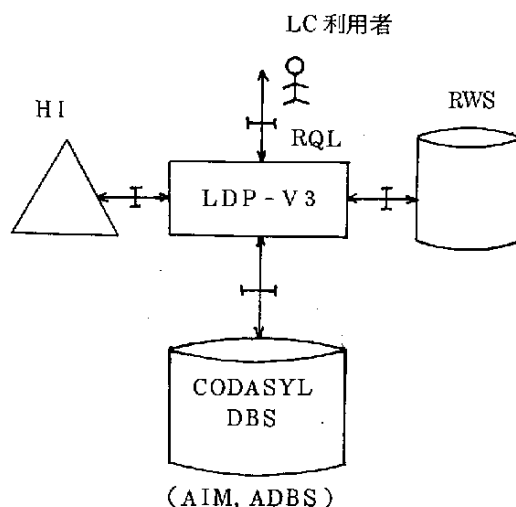


図8.1 LDP-V3の構成

LDPは、図8.2に示す様に、HIP(HI(heterogeneity information) processor)とQTP(query translation processor)とから成る。HIPは、CODASYL DBSのCODASYLスキーマ(LIS)から、等価な関係スキーマ(LCS)を生成し、異種性情報(HI)に、LCSと、LISとLCSとの対応情報を格納する。

QTPは、関係スキーマLCSに対する関係操作演算RQL(relational query language)を入力して、HIを用いてCOBOL DMLに変換し、CODASYL DBS上でこれを実行させる。RQL演算が検索のときには、COBOL DMLプログラムの実行結果を、関係としてデータベースから導出し、RWS(関係作業領域)に格納する。利用者は、RWS内に導入された関係を、編集した結果をLDPを通して表示できる。

HIとRWSは、1つのCODASYL DBSとして管理される。

LDP-V3の詳細は、第Ⅳ部の2章を参照されたい。

LDP-V3は、PL/Iでコーディングされ、ソース文数約15,000であり、実行形式は約250KBである。M-170Fで実働するLDP-V3は、OS IV/F4のもとで動作する。

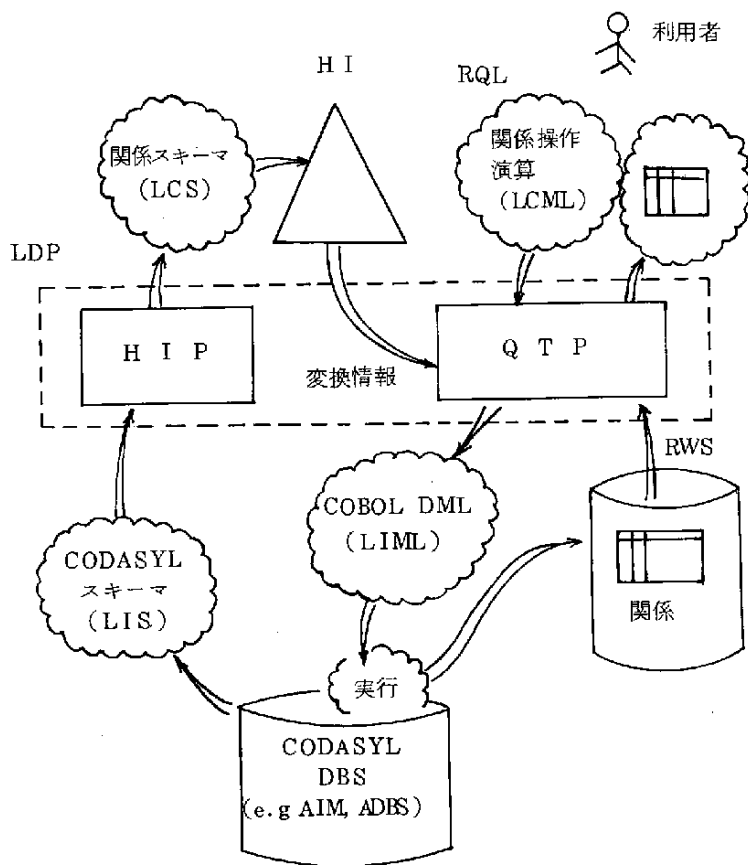


図8.2 LDPの概要

9. LDPの性能と利用性の評価

本章では、ローカルデータベースプロセッサ(LDP)の性能と利用性との2点についての評価結果について示す。性能評価では、利用者の入力したRQLによるLC問合せから、COBOL DMLを生成する為の経過時間をはかり、LDPのオーバーヘッドを明らかにするとともに、LDP内の各モジュール毎のオーバーヘッドを明らかにし、パフォーマンスの隘路を明らかにする。利用性評価では、利用者がLDPを通して必要な解を導出する為の全作業時間と、COBOL DMLによって直接解を得るときの全作業時間を明らかにする。両者の評価によって、LDPのオーバーヘッドが、利用者の得る全作業時間の短縮によって、十分に償なわれることを明らかにする。

9.1 性能評価

本章では、LDPの性能面からの評価について述べる。性能評価は、以下の点を目標にしている。

- (1) LDPの各構成モジュールの負荷を明らかにする。これから、性能面での隘路を求め、改良を行う。
- (2) アクセスパス生成の負荷と、それによって生成されたCOBOL DMLプログラムのアクセス効率を明らかにする。
- (3) 選択度、結合度といった統計情報の正しさの検証を行う。

9.1.1 評価環境

LDPを、以下の環境下で評価する。

- (1) LDPを、M-170F上で実働させる。Acos-700に対しては、M-170F上のLDPで生成されたCOBOL DMLプログラムをADBS上で動かす。
- (2) LDPを単一利用者の下で実働させ、その経過時間(elapse time)を測定する。親言語の時間取得機能(call CPTIME)で得た経過時間は、M-170Fでは10%から20%の誤差があるが、今回は各評価問合せに対して各一回実行させる事とした。
- (3) 評価用のデータベースとしては、第IV部の第1.3で示すプロジェクトデータベースシステム(PRDBS)を用いる。PRDBS内の各レコード型、セット型、リンクレコード型の実現値数とサイズを表9.1に示す。

9.1.2 各構成モジュールの負荷

図9.1に、第IV部2章で述べる各構成モジュールの負荷を、各々の経過時間で示す。図から分る様にDMLGでのCOBOL DMLプログラムを文字列として生成する為に全経過時間の70%から90%が費やされてしまう。アクセスパス生成の為の経過時間の割合は高々数%にしか過ぎない。又、QUEL問合せから内部表現としての木、グラフの生成と操作の為に10%から20%が費やされている。

LDPの性能を向上させる為に、DMLGの改良を行った。DMLGでは、DMLの文字パターン

表 9.1 PRDBS の実現値情報

関係性名	タイプ	実現値数	レコード型	実現値数
SYS-ORG	セット	1	EMPLOYEE	68
SYS-EMP		1	ORGANIZATION	15
SYS-SPON		1	PRODUCT	9
ORG-EMP		15	SUBJECT	136
SPON-PROJ		4	PROJECT	14
PROJ-PROD	リンク	9	REPORTR	84
E-P		62	SPONSOR	4
P-E		62		
E-R		155		
R-E		155		
R-S		625		
S-R		625		
P-S		201		
S-P		201		
PRIOR-PROJ		5		
POST-PROJ		5		

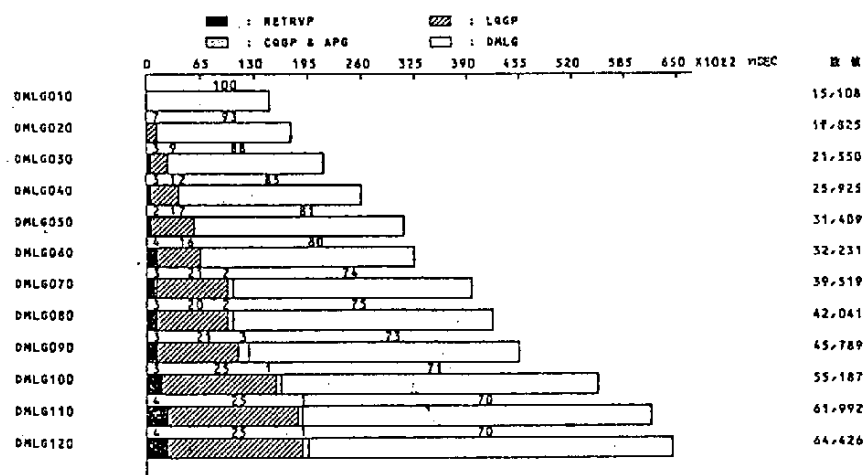


図 9.1 各構成モジュールの負荷

を一字ずつ左から右にサーチして、パラメータを見付けて、その置換を行っている。

例 `find next $01 within $02.`

→ `find next EMPLOYEE within ORG-EMP.`

この為に、以下の改良を行った。

- (i) パラメータを左づめする。
- (ii) これより変換の不要となる右端を明示する。
- (iii) コメント文を除去する。

これによって、図 9.2 に示す様に DMLG の経過時間を 30% から 40% 短縮できた。

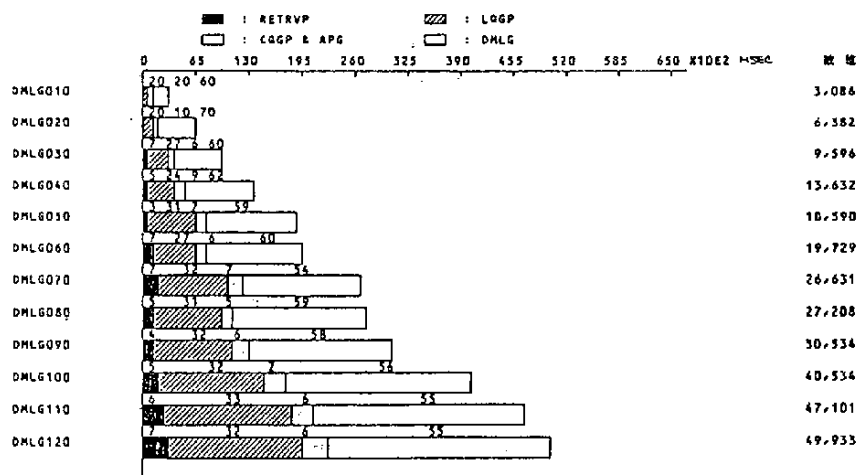


図 9.2 改良後の各構成モジュールの負荷

9.1.3 アクセスパス生成の負荷

アクセスパスは、DFA 1、2、3 の三種の発見法によって、CODASYL 問合せグラフ (CQG) から生成される。アクセスパス生成に必要な経過時間を、CQG の辺数に対して、プロットしたものが図 9.3 である。

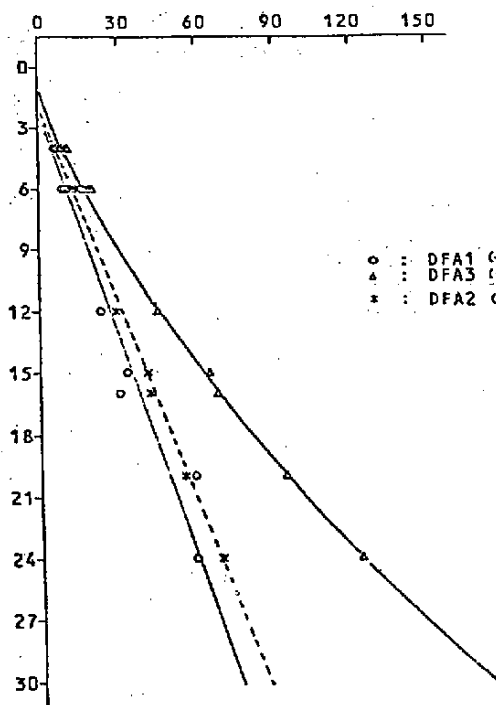
DFA 1 は、一レベルのサーチを行う発見法である。図から分る様にほぼ線形に経過時間が増加している事が分る。DFA 2 もほぼ線形に経過時間が増加している。一方 DFA 3 は、辺数に対して、二乗のオーダーで経過時間が増加している。

9.1.4 アクセスパス生成

アクセスパスは、生成された COBOL DML プログラムの全経過時間を短縮する為に、アクセスされるレコード型の実現値の期待値が最少となる様に、選択度、結合度といった統計情報を基にして生成される。ここでは、COBOL DML プログラムの経過時間がアクセスされる実現値数に単調増加するものと仮定している。図 9.4 は、種々の問合せから生成される COBOL DML プログラムについて、実際にアクセスされた実現値数と経過時間との関係を示している。図から分る様に、この仮定が正しい事が分る。

アクセスパスは、選択度、結合度といった統計情報を基にして、7 章で述べた様なコスト関数に基づいて見積られる。我々の用意した 8 個の問合せについて、見積りと、実際にアクセスされた実現値数とを表 9.2 に示す。この例では、見積りと実際の値との間に、あまり相関がみられない。

統計情報が、データベース全体の実現値についての平均値であるのに対して、個々の問合せは、デ



辺数

図 9.3 アクセスパス生成の負荷

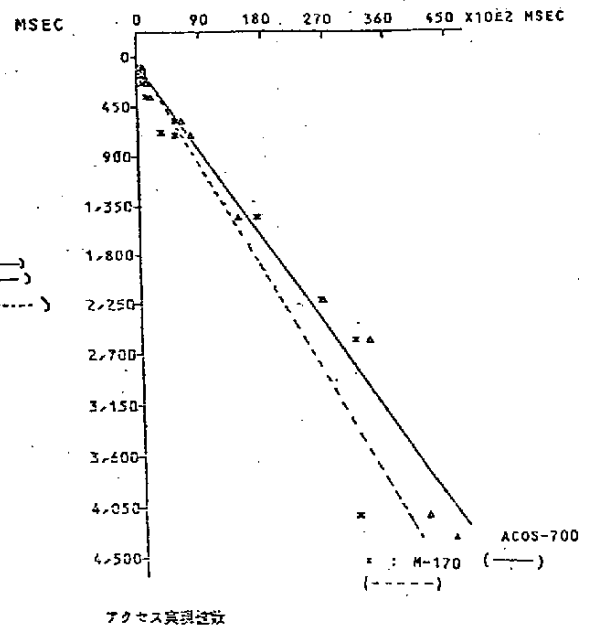


図 9.4 アクセスされた実現値数と経過時間

表 9.2 実現値数の見積りと実際

Query	No. of Node	No. of Join	DFA 1		DFA 2		DFA 3	
			Est.	Act.	Est.	Act.	Est.	Act.
10	5	4	107	231	107	231	107	231
20	7	6	158	6981	158	6981	158	6981
30	5	4	31	92	31	92	31	92
40	12	12	194	954	267	993	306	2281
50	15	16	503	19267	503	34672	889	18681
60	21	24	3163	22839	3163	51251	5875	33747
70	14	15	577	4932	396	2411	396	1791
80	18	20	577	4932	549	2411	54	12090

データベースの一部のみをアクセスする。このアクセスされる部分の値分布が均一でない為に、このような現象が生じたと考えられる。従って、

(i) 一般に、データベース全体の値分布は、各アプリケーションのアクセスする値分布と一致しない。

(ii) 各アプリケーションのアクセスするデータベースの範囲は、比較的限られている。

これらの事より、見積りと実際の実現値数との関連を密接にする為に、以下の様な方法を用いる。

(i) 統計情報をアプリケーション毎に備える。データベース内で、アプリケーションがアクセスする部分についての実現値の統計情報を用意する。

(ii) 結合度をより現実と適合させる為に、次の様な動的なチューニング方法を用いる。Sをセット型とする。 $ent(S)$ 、 $icnt(S)$ を各々Sの正、逆結合度とする。あるアプリケーションプログラムの実行について、 $oent(S)$ と $ment(S)$ を、各々Sの親から子にアクセスした時の親と子のアクセスされた実現値の総数とする。 $oicnt(S)$ と $micnt(S)$ を各々、子から親をアクセスした時の子及び親のアクセスされた実現値の総数とする。

$$ent(S) \leftarrow \text{if } ment(S) \neq 0, \text{ then} \\ \quad [ent(S) + (oent(S) / ment(S))] / 2$$

$$icnt(S) \leftarrow \text{if } micnt(S) \neq 0, \text{ then} \\ \quad [icnt(S) + (micnt(S) / oicnt(S))] / 2$$

図9.5は、動的チューニングによる見積りの改良を示している。図より、動的チューニングによって、結合度を現実の状態に近付ける事ができる事が分る。

図9.6は、制限式を含んだ場合を示している。データベース全体の統計情報による見積りよりも、

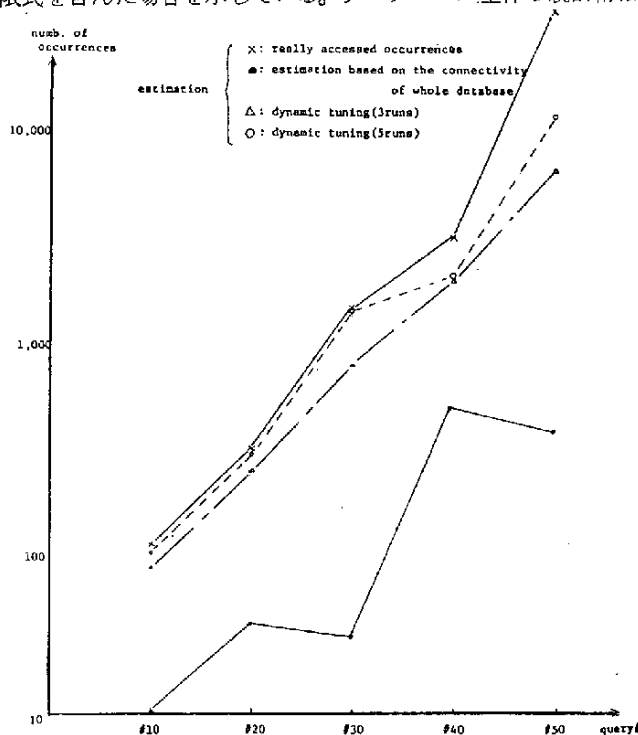


図9.5 動的チューニングによる見積りの改善

実際に近付いているが、図 9.5 ほどではない。これは、選択度がアクセスされたデータベースの範囲の状況を反映していない事によっている。CODASYL データベースシステムの選択度としては以下の点を考慮する必要がある。

- (1) レコード型の各項目の値毎に、この値を持つ実現値数を保持せねばならない。
- (2) レコード型全体と共に、各セット実現値内での子レコード実現値の値分布が必要になる。
- (3) 等制限式は、(1)及び(2)によって正しく求められるが、不等制限式($<$, $\leq$, $\geq$, $>$)に対する選択度を正しく求めるには、完全な値の分布が必要になる。

この為の情報を全て異種情報に持たすには、第 1 に格納空間の大きさが問題になる。(1)については、1つの方法は、よく用いる値に対してのみ情報を保つことである。しかし、(2)については、アクセスコストの見積りの時点でセット実現値内での制限式の見積度を、反映させることは出来ない。何故なら、あるセット型でアクセスされる子実現値は、実際にアクセスされて、はじめてわかるからである。

アクセスパスを見付ける為のサーチレベルの深さと、その効果について考える。ここでは、動的チューニングによって、統計情報が、実際の状態に合う様になっているとする。表 9.3 は、この時の 3つの方法 DFA 1、2、3 の見積り数と、実際のアクセス実現値数を示している。表から分る様に、

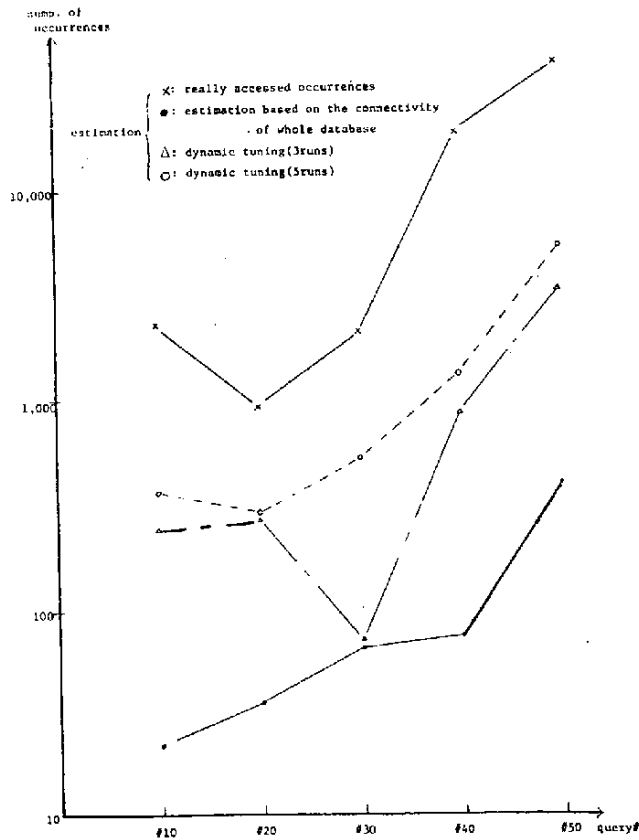


図 9.6 制限式を持った問合せに対する見積り

サーチレベルを深めても、あまり効果はない。

表 9.3 サーチレベルと効果

Query #	No. of Node	No. of Join	D F A 1		D F A 2		D F A 3	
			Est.	Act.	Est.	Act.	Est.	Act.
10	5	4	110	117	110	117	110	117
20	7	6	142	151	142	151	142	151
30	8	8	1903	2215	1903	2215	1903	2215
40	11	11	3799	1194	3799	1199	3801	1199
50	12	12	2318	3593	425	7585	3076	3335
60	13	14	3270	4697	6358	3343	6358	3343
70	14	15	8328	2623	5321	7656	5340	6190
80	18	20	824	3135	1563	1573	2423	8093

9.1.5 COBOL DMLプログラム

生成されたCOBOL DMLプログラムの実行結果について考える。COBOL DMLプログラムは、アクセスされるレコード型とセット型の数（CQGのノード数と主辺の数）に対して、表9.4に示す様なプログラム文数となる。

表 9.4 プログラム文数

QUERY #	NO. OF NODES	NO. OF JOIN	NO. OF COBOL STEPS		
			DATA DIV.	PROC. DIV.	TOTAL
R1	5	4	103	358	461
R2	8	8	112	426	538
R3	7	6	112	384	496
R4	16	15	137	594	731
R5	12	12	123	530	653

COBOL DMLプログラムの実行は、次の3部分から成る。

- (1) 初期化と終了
- (2) 実現値の巡航
- (3) 出力処理

これらをまとめると表3.5のようになる。表から分る様に、初期化/終了の為の経過時間は、ほぼ同

一で次の様である。

表 9.5 COBOL DMLプログラムの経過時間

QUERY #	DBMS	COBOL ELAPSE TIME (MSEC)		
		INIT/END	NAVIGATION	OUTPUT
10	AIM	610	7220	250
	ADBS	7210	3730	430
20	AIM	640	195950	5860
	ADBS	6890	195840	37400
30	AIM	650	411240	13630
	ADBS	6750	618180	95850
40	AIM	650	45480	1200
	ADBS	6670	43350	8120
50	AIM	670	62940	1760
	ADBS	6660	10100	1800

AIM ~ 650 [ms]
ADBS ~ 6800 [ms]

9.2 利用性評価

次に、LDPの利用者から見た有用性について評価する。これは、以下の2つの場合の全ライフサイクルの経過時間を測定し、前者の有用性を経過時間として定量化する事を行う。

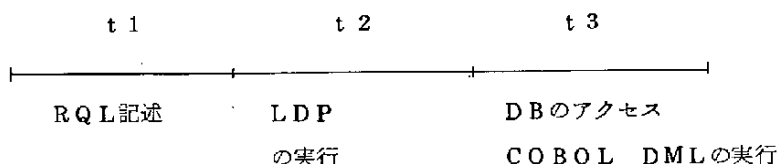
- (1) RQLによって問題を記述し、これをLDPでCOBOL DMLプログラム変換し実行させ、結果を得る。
- (2) 同一の問題に対して、COBOL DMLを用いて、設計し、コーディングし、デバッグして、結果を得る。

9.2.1 評価方法

評価を次の様にして行う。

- (1) 同一の日本語の問題に対して、以下を行う。
 - (i) 計算機の素人によって、LCSに基づいて問題をRQLで記述する。
 - (ii) COBOLエキスパート(経験10年)によって、CODASYLスキーマに基づいて、問題をCOBOL DMLで記述する。
- (2) M-170FのAIM上のPRDBSを用いる。
- (3) RQL及びCOBOL DMLによる問題解決のライフサイクルの各ステップの経過時間を測定する〔図9.7〕。

RQL



COBOL DMLプログラム

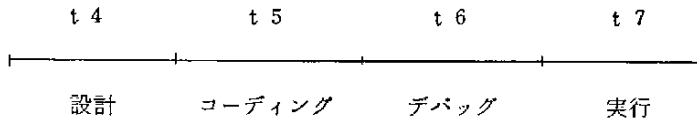


図 9.7 ライフサイクル

9.2.2 問 題

問合せ問題を、以下に示す。

- R1 各 project のメンバの書いたレポートの中で、このメンバが第一著者 (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 のメンバの番号 (eno) と、その論文番号 (rno) を求めよ。
- R3 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 (hired-year) したメンバの名前 (ename)、番号 (eno)、及びその project 番号を求めよ。
- R4 distributed database systems と、database systems と、four-schema 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 番号及び、このメンバが第一著者である全ての論文の番号を求めよ。

9.2.3 評 価 結 果

表 9.6 に、測定結果を示す。

表 9.6 利用性評価結果

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=RQL、(H)=HOUR、(M)=MINUTE、(S)=SECOND

表から分る様、LDPでは、数分から数10分で結果を得れる。しかし、COBOLによつては、結果を得るまでにエキスパートでも数時間から数10時間を要してしまう。この意味で、CODASYLデータベースシステム上に、非手続的な検索インタフェースを設ける事は、ソフトウェアの生産性の点から優れていることが分る。

アクセスパス生成では、簡単な問題に対して、LDPはエキスパート同様なアクセスパスを生成できるが、複雑な問題に対しては、エキスパートのものより劣っていることが分る。又、LDPによつて生成されるCOBOL DMLプログラムの量でもエキスパートより数倍多くなっている。

アクセスパス生成でエキスパートとLDPとの差は、特にCQGがループを含む問合せにおいて、アクセス方法が異なる。LDPは、なるべく途中で中間結果を生成せずに一連の巡航アクセスによつて結果を得ようとする。この為、合流節点等に於ける現在子の保持方法に注意を払っている。しかし、エキスパートは、むしろループを分割して中間結果を生成し、これを最後にまとめて処理しようとしている。

9.3 まとめと今後の課題

本章では、CODASYLデータベースシステム上の非手続的な関係インタフェースシステム、ローカルデータベースプロセッサLDPの性能及び利用面からの総合的な評価について述べた。

LDPは、現在、当協会のM-170FのAIM及びAcos-700のADBS上に、PL/Iによつて実現されている。

性能面の評価では、まずLDPの各構成モジュールの負荷を経過時間によつて測定し、隘路となるモジュールを明らかにした。隘路となるモジュールは、COBOL DMLプログラム文字列の生成を行うDMLGであり、全経過時間の過半数を占めていた。これの改良によつて、全体に、20%~30%経過時間を短縮出来た。アクセスパス生成は全体の高々数%の負荷であり、LDP全体から見ると無視し得る程度である。現在、DMLGと共に、RQLによるLC問合せからCODASYL問合せグラフ(CQG)を作るまでの負荷が大きな重みを占めている。ここでは、木、グラフ等の構造の非手続的な操作が行われるからである。

集積関数処理の評価については付記5を参照されたい。

10. まとめと今後の課題

第Ⅱ部では、異種分散型データベースシステムJDDBSにおける異種性問題の解決法を示した。まず、CODASYL DBSの階層を明らかにし、各層の論理的な形義を与えた。CODASYL DBSを概念、論理、仮想、物理の4層からとらえ、各層毎にデータ構造と、この上の操作演算を明らかにした。まず論理層では、実体の集合と、2つの実体集合間の部分関数を表す関連集合とから成る論理データ構造と述語論理に基づいた非手続的なCODASYL問合せ言語(CQL)及び、そのグラフ表現としてのCODASYL問合せグラフ(CQG)とを定義した。CQLでは、検索(問合せ)演算と、更新演算とを定義した。CQGは、問合せの意味の直観的理解を与えられるので、CODASYL DBS上の一般利用者用の論理的な操作言語となるものと考えている。又、部分関数の型としてのメンバシップクラスによる基本更新演算の意味を定めた。仮想層では、論理データ構造に、順序関係を導入した仮想データ構造を定義し、仮想データ構造上のCOBOL DML演算を実行出来る仮想的なCODASYLマシン(VCM)を、多テープのチューリングマシンとして表した。これによって、CODASYLモデルのDMLの意味を明らかに出来たと考える。仮想層は、CODASYLモデルの物理的側面を除いた層である。物理層では、CODASYLモデルにおける物理的な仮想データ構造の格納方法と、アクセス方法とを示した。更に、概念層では、論理データ構造における実体集合間の部分関数に加えて、 n (≥ 2) 個の実体集合間の一般の n 項関係を含した概念データ構造を定義した。

概念層のデータモデル(データ構造と操作演算)を、関係表現したローカル概念(LC)モデルを定めた。概念CODASYLデータ構造とLCデータ構造との1対1対応性と、操作(更新)演算の1対1対応性を示し、両モデルの等価性を明らかにした。このLCモデルは、実体集合(E関係)と、実体集合間の任意の関連集合(R関係)とからなる各データモデルを概念化した概念モデルの関連集合を型によって特性化し、型による更新演算の意味を表したモデルでもある。よって、概念モデルの関連集合を、LCモデルのT関係だけに限定し、関連による閉路のない木型のスキーマだけを許すモデルは、IMSモデルともなる。

次に、LCデータ構造上の述語形式の関係操作言語RQLによる問合せを、論理CODASYLデータ構造上のCQL問合せによって表せることを示した。

CQL問合せは、述語形式の非手続的問合せであるので、これをVCMで実行できるDML演算で表せねばならない。この為、まず全てのCQLを、DMLで表せることを示し、ついて、出力テープ(ファイル)を最少化するDMLの生成方法を示した。本報告書で示した方法では、唯1つの出力用の順次ファイル上への順次書込みだけで、目標集合を生成出来る。但し、冗長性は除去されていない。この方法が、正しいことを論理的にも示してある。同時に、DMLのアクセスコストを、アクセスされる実現値数としたときのアクセスコストの期待値を求めている。

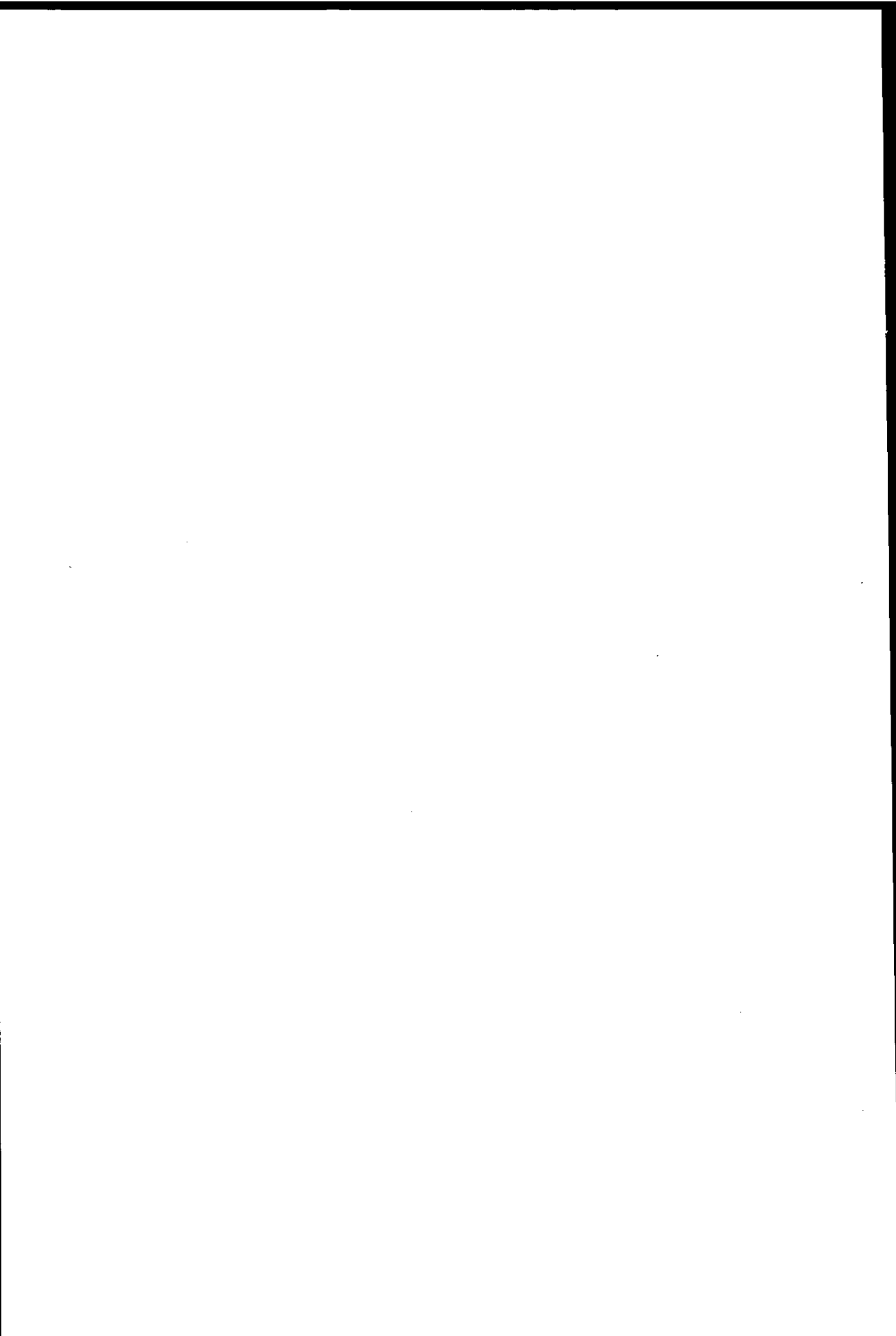
以上によって、CODASYL DBS上に、関係インタフェースを提供する為の論理的な基礎を

示せた。本部で述べた概念に基づいたローカルデータベースプロセッサ(LDP-V3)は、既に当協会のAIM(M-170F)、ADBS(Acos-700)上で実働し、当協会の業務に用いられている。

今後、次の検討を行う予定である。

- (1) LC問合せグラフ(LQG)、CODASYL問合せグラフ(CQG)による図形による問合せ記述インタフェースの設置。
- (2) アクセスパスのより有効な最適化方法、出力コストと、アクセスコストとの関連の明確化。
- (3) エディタと一体化した利用者インタフェースの設置
- (4) LDPを通してのCODASYL DBSの定義(スキーマの設計、DBの定義)
- (5) トランザクション(複数RQL問合せ/更新)の記述とDML化。
- (6) COBOL DMLプログラムの開発ツール
- (7) PL/I DML、FORTRAN DMLへの対応

第Ⅲ部 分散性問題



第Ⅲ部 分散性問題

分散型データベースシステム (DDBS) が、複数のデータベースシステム (DBS) を通信網を用いて相互結合して構成されることから生じる分散性問題としては次の2点がある。

(i) 設計問題

(ii) 分散検索／更新問合せ処理

(i)の設計問題としては、ローカル概念 (LC) データ構造と、全体概念 (GC) データ構造との間の写像をどのように行うかである。統合的設計方法では、各DBS上に、DDBS全体で共通なデータモデルに基づいたLCデータ構造がまず定義されていて、これらのLCデータ構造上に、GCデータ構造が定義される。逆に分割型設計方法では、まずGCデータ構造が定義されて、その後に全体の利用者の利用要求に会う様に、GCデータ構造を各データモジュール (DM) に分割してLCデータ構造とされる。我々のJDDBSでは、前者の統合的設計方法を用いている。

(ii)では、GCデータ構造に基づいた演算 (全体概念 (GC) 演算) を、どの様にLCデータ構造上の演算と、各データモジュール (DM) 間の通信処理とによって表すかが問題となる。GC演算としては、検索と更新演算とがある。検索演算の分散処理では、各DM内のローカル処理とDM間での通信処理コストをいかに最少化していくかが問題になる。通信網の速度、信頼性、通信形態 (1対1通信又は1対多 (放送) 通信) が、分散検索処理を考える上で重要である、我々のJDDBSでは、放送通信に基づいた検索の処理方式を用いている。更新演算の分散処理では、各トランザクションの複数のDMへの更新演算の原子性を保つ為のコミットメント制御と (Commitment control)、各DMへの複数のトランザクションからの検索と更新演算の実行順序をなるべく並行度を高めながらかつDDBS全体のインテグリティを保つ為の同時実行制御 (concurrency control) とが必要である。

更に、セキュリティ、障害回復の制御機能も必要となる。JDDBSではこれらの問題のうち、コミットメント制御方法について、放送通信と各DMの分散制御に基づいた方法を用いている。

1. 統 合 化

統合化とは、各データモジュール (DM) 内のローカル概念 (LC) データ構造上に、DDBSのある共同体にとって必要な一意な全体概念 (GC) データ構造を定義し、GCデータ構造とLCデータ構造との対応情報を分散情報 (DI) として生成する過程である〔図 1.1〕。

GCデータ構造の定義は、DDBSの全体管理者 (GA) が行う。

1.1 GCデータ構造

全体概念 (GC) データ構造 G は、GC関係スキームの集合としてのGCスキーマ (GCS) と、 G 内の各GC関係スキームに実際に値を与えたGC関係の集合としてのGCデータベース G とから成る。 G 内の各GC関係スキーム R は、属性集合 $\Omega_R = \{ a_1, \dots, a_m \} \ (m \geq 1)$ とインテグリティ条件 F_R とから成る。 $(R = (\Omega_R, F_R))$ 各属性 $a_i \in \Omega_R$ の取り得る値の集合を定義域とし、 $\text{dom}(a_i)$ と表す。 F_R を省略して、GC関係スキームを $R(a_1 \llcorner \text{dom}(a_1), \dots, a_m \llcorner \text{dom}(a_m))$ と表すことにする。

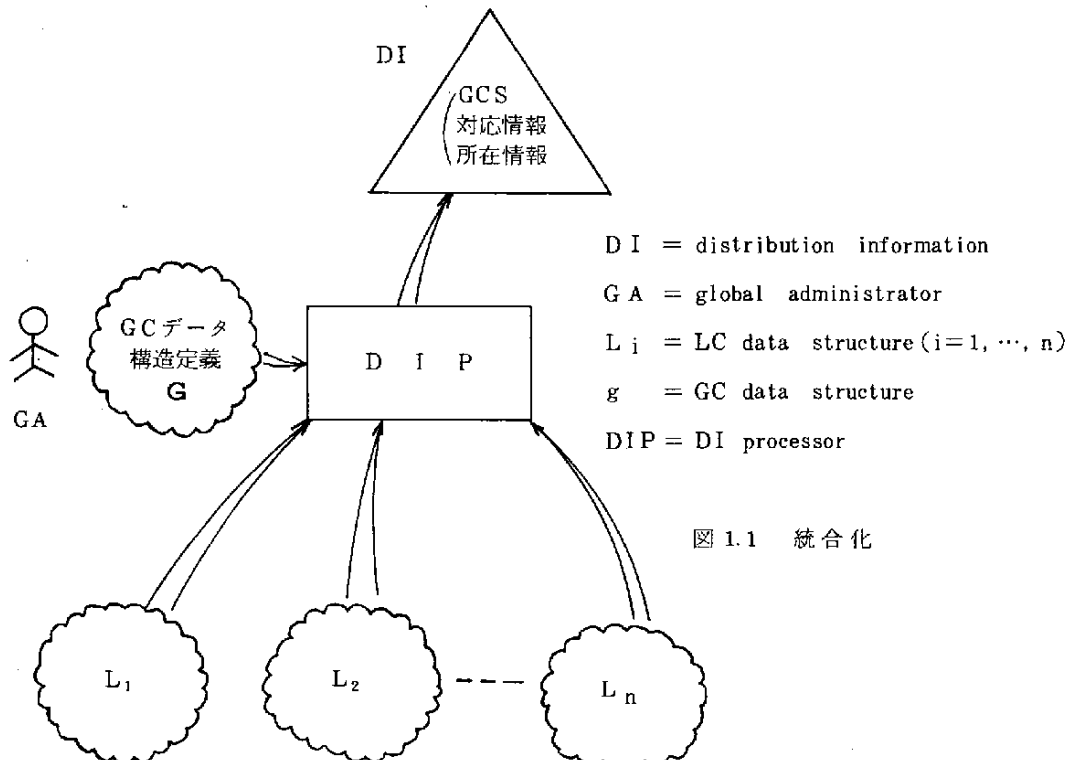


図 1.1 統合化

各 R に実際に値を与えたものをGC関係 $R = \{ r \mid r \in \text{dom}(a_1) \times \dots \times \text{dom}(a_m) \wedge F_R(r) \}$ とする。各要素 $r \in R$ を、組 (又はGC組) と呼び、 r の属性 $a_i \in \Omega_R$ の値を $a_i(r)$ と表す。

GCデータ構造は、通常の関係データ構造である。

1.2 GC 演算

GC データベースに対する検索演算を、ローカル概念 (LC) 層と同じ構文の関係述語言語 RQL によって表す。RQL は、QUEL と同様の構文を持ち、次の様に記述される。

range (g_1, G_1) (g_l, G_l) ;
retrieve into T (tl) where qual ;

ここで、 G_i は全体概念 (GC) スキーマ内の GC 関係スキーム名である ($i = 1, \dots, l$)。

range 文は、問合せの参照する各 G_i に対して、GC データベース内の GC 関係 G_i の組を取る組変数 g_i を定義する。問合せ内で G_i の属性 a は、 $g_i.a$ と表される。

T は、利用者の必要とする目標関係スキーム ($I = (\Omega_T)$, $\Omega_T = \{a_1, \dots, a_k\}$) 名である。 tl は目標リストと呼ばれ、次の様に書かれる。

$tl ::= a_1 = \text{exp}_1, \dots, a_k = \text{exp}_k$
 exp_j は、定数、属性、又はこれらの上の算術式である。 tl は、 $G_1 \times \dots \times G_l$ から T への関数と考えられる。ここで、 $G_i = \text{dom}(\Omega_{G_i})$, $T = \text{dom}(\Omega_T)$ である。

qual は、条件式 (qualification) と呼ばれ、次の様に積正規化された論理式である。

$\text{qual} ::= C_1 \wedge C_2 \wedge \dots \wedge C_n$
 $C_i ::= r_{f_i} \mid j_{f_i} \quad (j=1, \dots, n)$
 $r_{f_i} ::= r_{p_{i1}} \vee \dots \vee r_{p_{ih_i}} \quad (h_i \geq 1)$
 $j_{f_i} ::= j_{p_{i1}} \vee \dots \vee j_{p_{ip_i}} \quad (p_i \geq 1)$
 $r_{p_{ij}} ::= \text{exp}_j \quad \theta \quad v$
 $j_{p_{ij}} ::= \text{exp} \quad \theta \quad \text{exp}$

ここで、 $\theta \in \{<, \leq, =, \geq, >, \neq\}$ であり、 v は定数である。 $r_{p_{ij}}$ は、制限述語であり、 r_{f_i} は、同一の組変数に関する制限述語の和である。 $j_{p_{ij}}$ は、結合述語であり、 j_{f_i} は、変数に関する結合述語の和で、結合式と呼ばれる。 r_{f_i} は変数 g 、 j_{f_i} は変数 g_j 、 g_h に関する式とすると、各々は次の様に考えられる。

$r_{f_i} : G \rightarrow \{\text{true}, \text{false}\} \quad (G = \text{dom}(\Omega_G))$

$j_{f_i} : G_j \times G_h \rightarrow \{\text{true}, \text{false}\} \quad (G_j = \text{dom}(\Omega_{G_j}), G_h = \text{dom}(\Omega_{G_h}))$

算述式 exp_i は、集積関数 (aggregate function) を含んでもよい。以上より条件式 qual は次の様に考えられる。

$\text{qual} : G_1 \times \dots \times G_l \rightarrow \{\text{true}, \text{false}\}$

ここで $G_i = \text{dom}(\Omega_{G_i}) \quad (i = 1, \dots, l)$

GC 関係 $G_1, \dots, G_l$ に対する問合せの意味は、次の様である。

$T = \{t \mid t = tl(g_1, \dots, g_l) \wedge_{i=1}^l g_i \in G_i$
 $\wedge \text{qual}(g_1, \dots, g_l)\}$

JDDBSでは、GC問合せに以下の制限を設ける。

- (1) 結合式は、全て等結合だけから成るとする。
- (2) group-by 集積関数及び単純集積関数内に、group-by 集積関数を含んではならない。
- (3) 問合せは算術関数 (e.g., exp, log, sin) を含まない。

1.3 GCデータ構造定義

GCデータ構造Gは、各データモジュールDM_iのLCデータ構造L_i (i = 1, ..., n)上の視野として定義される。よって、GCデータベースgは、仮想的となる。ここで、LをLCデータ構造の集合とする (L = { L₁, ..., L_n })。LをLCスキーマの集合とし ($\mathcal{L} = \{ \underline{L}_1, \dots, \underline{L}_n \}$) $\mathcal{L}$ をLCデータベースの集合とする ($\mathcal{L} = \{ \mathcal{L}_1, \dots, \mathcal{L}_n \}$)。GCデータ構造の定義式δは、 Ω_{Rij} (L_{ij} ∈ $\mathcal{L}_i$) によって表され、δは、 $\mathcal{L}_1 \times \dots \times \mathcal{L}_n$ からGへの関数である。GCデータ構造内の各GC関係スキームは、次のGCデータ構造内の各GC関係スキームは、次のGCデータ構造定義言語 (GCDL) によって定義される。

range (l₁, L₁ : d₁) (l_p, L_p : d_p) ;(1)

define G (a₁, ..., a_m) <def> ;(2)

ここでL_iは、LCスキーマd_i内のLC関係名で、l_iはその組変数である。組変数の定義域は、LC関係L_iで、値としてL_i内の組を取る。(1)のrange文は、LC関係L_iのLC組変数l_iを定義している (i = 1, ..., p, p ≥ 1)。

(2)のdefine文は、LC関係の直積L₁ × × L_pから、スキーマG (a₁, ..., a_m) なるGC関係Gへの関数を定義している。<def>は、次の様である。

<def> ::= <subdef> | <subdef> <sop> <def> | (<def>)

<sop> ::= U | ∩ | -

<subdef> ::= t1 where qual

t1 ::= (a₁ = exp₁, ..., a_m = exp_m)

ここで、exp_iは、定数、LC属性又はLC属性上の算術式である。t1は、関数L₁ × ... × L_p → G_i (⊆ G) と考えられる。qualは、次の様に積正規化された論理式である。

qual ::= C₁ and and C_h

c_i ::= p_{i1} or or p_{i1i} (i = 1, ..., h)

p_{i1} ::= exp θ exp

ここで、 $\theta \in \{<, \leq, =, \geq, >, \neq\}$ である。 exp は、定数、LC属性、又はLC属性上の算術式である。 c_i 内の各 p_{ij} ($j=1, \dots, l_i$)は、皆同一のLC変数を参照してはいるものとする。 p_{ij} が $\text{exp} \theta$ 又は、 $\text{exp}_1 \theta \text{exp}_2$ (ここで exp_1 と exp_2 は、あるLC変数1のみを参照している)であるとき、制約述語と呼び $p_{ij}: L \rightarrow \{\text{true}, \text{false}\}$ ($L = \text{dom}(\Omega_L)$)なる述語である。又、 p_{ij} が、 $\text{exp}_1 \theta \text{exp}_2$ で、 exp_1 と exp_2 の参照するLC変数が各々、 l_1 と l_2 で $l_1 \neq l_2$ のとき、これを結合述語 $L_1 \times L_2 \rightarrow \{\text{true}, \text{false}\}$ ($L_i = \text{dom}(\Omega_{L_i})$)と呼ぶ。又、各 exp は集積関数を含んでもよい。集積関数については第IV部の2章を参照されたい。

以上、 qual は、述語 $L_1 \times \dots \times L_p \rightarrow \{\text{true}, \text{false}\}$ と考えられる。よって、1つのsubdefの意味は、 $G_i = \{g_i \mid g_i = t1(l_1, \dots, l_p) \wedge (l_1, \dots, l_p) \in L_1 \times \dots \times L_p \wedge \text{qual}(l_1, \dots, l_p)\}$ となる。

(1)の定義式は、各subdefの G_i に、集合演算 $\langle \text{sop} \rangle \in \{\cup, \cap, -\}$ を適用した結果を、 $\underline{G}(a_1, \dots, a_m)$ のGC関係とすることを表している。

例として、図1.2の様に、3つのDM内に関係が分散しているとする。この時、次のGCS関係が定義される。

```

range ( e1, EMP:1 ) ( pe, PEL:1 ) ( p, PROJ:1 );
range ( e2, EMP:2 ) ( er, ERL:2 );
range ( r, REP:3 );
define ER1 ( e2. @E, e2. eno, e2. ename, r. rno, r. title )
where
    e2. @E = er. @E and er. auth-no = 1 and
    er. @R = r. @R ;
define EP ( e1. @E, e1. eno, e1. ename, p. pno )
where
    e1. @E = pe. @E and pe. @P = p. @P ;

```

} (2)

ER1は、従業員と彼を第1著者とする論文情報から成るGCS関係である。PEは、従業員と彼の属しているプロジェクトの情報の関係である。図1.3は、GC関係ER1とEPとの定義式のグラフ表現である。

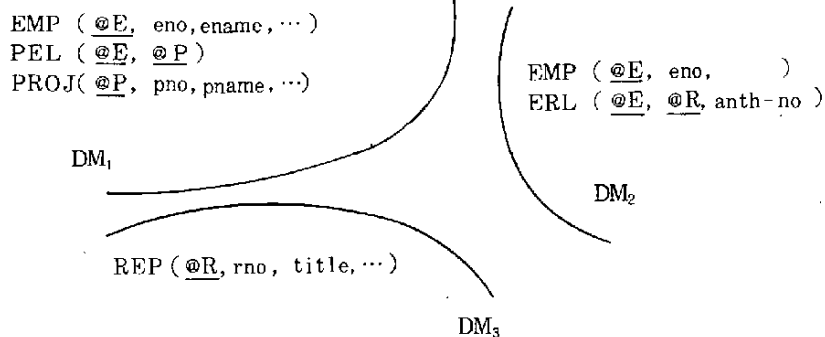


図 1.2 例

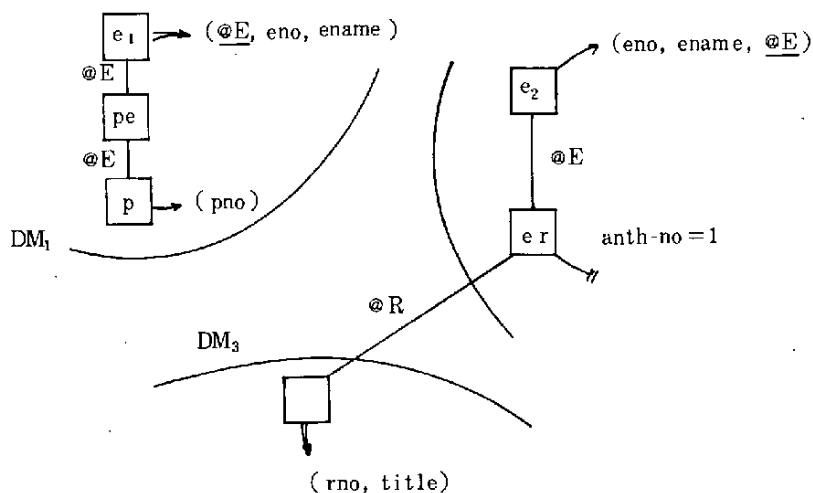


図 1.3 GCS 定義例

1.4 分散情報 (D I)

分散情報 (D I) は、以下の情報から成る。

- (1) G C データ構造 G の定義
- (2) G C データ構造と、L C データ構造との対応関係

詳細は、第IV部を参照されたい。

2. 分散問合せ処理 (DQP)

分散問合せ処理 (DQP) とは、全体概念 (GC) データベース上の検索演算を、ローカル概念 (LC) データベース上の演算によって表し、各DM内/DM間の通信処理によって解を求めることである。GCデータ構造上の検索演算をGC問合せ (GQ) とし、LCデータ構造上の検索演算をLC問合せ (LQ) とする。DQPは、次の3つの主要なステップから成る。

- (1) GC問合せ (GQ) を、LC関係を参照する問合せで表す。これを全体LC問合せ (GLQ) と呼ぶ。この処理を、表現変換と呼ぶ。
- (2) GLQを、各データモジュール (DM_i) で独立に処理出来る部分をLC問合せ (LQ_i) として分割し、対応するDM_i でLQ_i を実行させる。これを、初期ローカル問合せ処理 (ILQP) と呼ぶ。
- (3) ILQP後のGLQは、DM間の結合から成る。これをDM内/間の通信処理によって処理して、GQの解を得て、利用者に出力する。これを、DM間問合せ処理 (DMQP) と呼ぶ。

我々は、(1)では、問合せ修正手法 [STONM76] を用いる。(2)のILQPは、(3)のDM間通信処理 (DMQP) に必要なデータのみを各DMが導出するために必要 [HEVNA78] であるが、むしろ異種DBSから成る分散型データベースシステム (DDBS) では、各DBSの異種データ構造から、DDBSで共通な関係データ構造を導き出す為に必要であると考えている。(3)のDMQPでは、Ethernet, 無線網の様な通信媒体を共有した通信網が提供する1対多 (放送) 通信機能に基づいた完全分散制御と動的通信処理スケジュール決定を備えた方式について述べる。

2.1 表現変換 (ETP)

ETPは、GCS関係に対するGCS問合せを、LCS関係を参照する全体LCS問合せ (GLQ) に変換する。GCS問合せ (GQ) も、GLQとともに、RQLによって記述される。GQからGLQへの変換は、DIのGCS関係式を用いて、問合せ修正 (query modification) [STONM76] 手法を用いて行われる。

例として、(2)式のGCS関係ER1とEPに対して、GCS問合せ「DDBSプロジェクトのメンバの論文を求めよ」を考える。これは、RQLで次の様に書ける。

```
range (er, ER1) (ep, EP);  
retrieve into T (ep. ename, er. title)  
where  
    er. @E = ep. @E and ep. pno = "DDBS"; ..... (3)
```

(3)式のGQは、グラフ表現 (RQG) すると図2.1の様になる。問合せ修正は、次の様にして行

われる。

- (i) GQ内のGCS関係の属性(GCS属性)を、(2)式のGCS定義内の目標リストを用いて、LCS属性の式に置換する。例えば、(3)式の $ep.pno$ は、 $p.pno$ に置き換える。
- (ii) GQの参照する全てのGCS関係をみつけ、その定義式の条件式 $qual$ を、GQの条件式に接合する。(3)で例えば、GCS関係に対して、条件式 $e1.@E = pe.@E$ and $pe.@P = p.@P$ を接合する。

この様にして、GQ(3)は、(4)式の全体LCS問合せ(GLQ)のようになる。

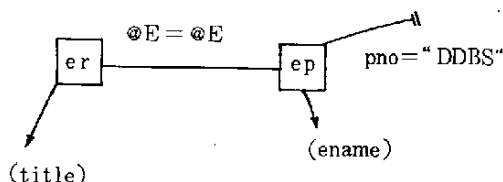


図 2.1 GCS 問合せ

```

range   (e1, EMP:1) (pe, PEL:1) (p, PROJ:1);
range   (e2, EMP:2) (er, ERL:2);
range   (r, RE:3);
retrieve into  T (e1.ename, e2.title)
where
    e2.@E = e1.@E and  p.pno = "DDBS" and
    e2.@E = er.@E and  er.auth-no = 1 and
    er.@R = r.@R and
    e1.@E = pe.@E and  pe.@P = p.@P ; ..... (4)

```

(4)式を、グラフ表現すると図 2.2 の様になる。図 2.1 からわかる様に、GQ が分散性と独立な問合せであるのに対して、GLQ は分散性が陽に現われた問合せである。

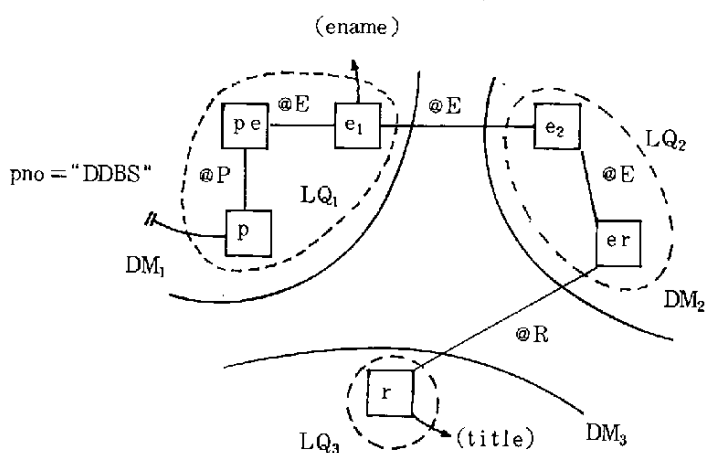


図 2.2 GLQ の RQG

2.2 1 LQP

QDPは、GLQを入力として、GLQを各DMで独立して処理出来るLCS問合せに分割する。これを問合せ分割と呼ぶ。DMで独立して処理出来る部分としては、次のものがある。

- (i) 同一のDM内の関係間の結合、
- (ii) 関係に対する制限、及び
- (iii) GLQの目標属性と、DM間の結合属性についての射影。

図 2.2 の例では、点線で囲んだ3つのLCS問合せ (LQ) に分割される。

```

LQ1 : range (e1, EMP:1) (pe, PEL:1) (p, PROJ:1);
       retrieve into T1 (e1.ename, e1.@E)
       where p.pno = "DDBS" and
       p.@P = pe.@P and pe.@E = e1.@E ;

LQ2 : range (e2, EMP:2) (er, ERL:2);
       retrieve into T2 (e2.@E, er.@R)
       where e2.@E = er.@R ;

LQ3 : range (r, REP:3);
       retrieve into T3 (r.title, r.@R);
  
```

LQ₁, LQ₂, LQ₃の実行は、各DM_iのLDP_i (i = 1, 2, 3)によってなされ、CO DASYLデータベースから、関係がRWS_iに導出される。これを初期ローカル問合せ処理 (I

LQP)とする。ILQP[HEVNA78]は、後述するDM間問合せ処理における通信量を減少させることも目的であるが、異種のDDBSではむしろ、異種のデータ構造から必要なデータを関係として導出する目的をもっている[TAKIM82c]。

この意味で、ILQPは異種DDBSでは本質的である。図2.3に、ILQP後のGLQを示す。

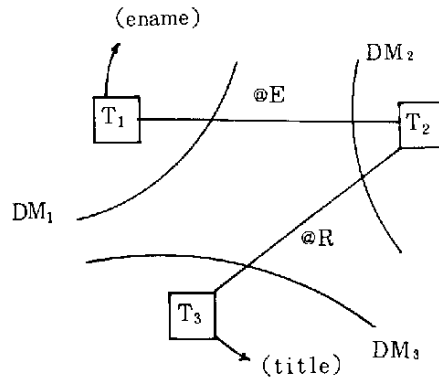


図 2.3 ILQP 後の GLQ

2.3 DM間問合せ処理 (DMQP)

データモジュール (DM) 間問合せ処理 (DMQP) とは、初期ローカル問合せ処理 (ILQP) 後のDM間問合せ (DMQ) を、問合せ処理に必要なデータを有するデータモジュール (DM) 内及びDM間の通信処理によって解くことである。我々のDMQPは、Ethernet, 無線網等の通信媒体を共有する通信網が物理的に提供する1対多(放送)通信機能を利用している。放送網では、あるデータモジュールDM_iが発信した信号(量x)は、網に結合された全てのデータモジュールDM_{ih}(h=1, ..., n)によって同時に受信出来る。このときの全通信時間C_i{j₁ ..., j_n}(x)と応答時間R_i{j₁ ..., j_n}(x)とは、同じであり、かつ全通信量xにのみ依存する。即ち、

$$C_i\{j_1, \dots, j_n\}(x) = R_i\{j_1, \dots, j_n\}(x) = a + bx$$

ここで、aとbは定数である。この性質を利用すると各DM_iが現在の状態情報を発信すれば、他の全てのDM_j(j≠i)は、このDM_iの状態を知ることが出来る。このことは、DMQPの各時刻において、DDBSを構成する各DMがDDBS全体の状態情報を持ててを示している。更に、各DM_jは、自分の有するこの全体状態情報を用いて、独自に、しかし同一の決定を各時刻で行えることも示している。以上より我々のDM間問合せ処理 (DMQP) は、次の特徴を有している。

- (1) DM間の結合処理の為の関係の放送。
- (2) 転送データ及び各関係のコード化によるデータ圧縮技法の利用。
- (3) 完全分散制御。
- (4) 動的決定。

2.3.1 DM間問合せ (DMQ)

初期ローカル問合せ処理 (ILQP) 後のデータモジュール間問合せ (DMQ) は、次の様な

形式を有している。

range ($r_1, R_1:d_1$) ($r_n, R_n:d_n$);
retrieve into $R(tl)$ where $qual$; } (1)

ここで、 r_i は、データモジュール $DM\ d_i$ 内の関係 R_i の組変数である。 tl は、目標属性の集合である。関係 R_i の目標項目の集合を tl_{R_i} とすると、 $tl = \bigcup_{i=1}^n tl_{R_i}$ である。
 $qual$ は、次の様に DM 間等結合式の積から成る。

$qual ::= jp_1 \text{ and } \dots \text{ and } jp_l$
 $jp_i ::= x.a = y.a$

$qual$ は、等結合式の推移関係から導ける全ての等結合式を含むものとする。

DM 間問合せ(1)内の属性を再名前づけする。まず、結合式 $x.b = y.c$ から推移関係で導かれる全ての結合式の結合述語の名前をある一意名 a_i とする。即ち、 $x.a = y.a$ とする。この時、以下を定義する。

各問合せ Q に対して

$\Sigma \triangleq \{ R_1, \dots, R_n \}$
 $\triangleq Q$ の参照する関係の集合。
 $TL \triangleq Q$ の目標項目集合。
 $A \triangleq Q$ 内の結合属性の集合。

各関係 $R_i \in \Sigma$ に対して

$\Omega_{R_i} \triangleq R_i$ の属性集合 ($\Omega_{R_i} = D_{R_i} \cup A_{R_i}$)。
 $D_{R_i} \triangleq R_i$ の存在するデータモジュール (DM)。
 $A_{R_i} \triangleq R_i$ 内の結合属性の集合。
 $TL_{R_i} \triangleq R_i$ 内の目標属性の集合。

各結合属性 $a \in A$ に対して

$R_a \triangleq \{ R \mid R \in \Sigma \wedge a \in A_R \}$
 $\triangleq$ 結合属性 a を持つ関係 R の集合

2.3.2 単純問合せ

データモジュール (DM) 間問合せ (DMQ) として、単純問合せについて考える。単純問合せとは、異なったデータモジュール (DM) 上の関係 $R_1, \dots, R_n$ ($n \geq 1$) 間に1つの結合属性 a についての等結合の積から成る問合せである。単純問合せ Q の問合せグラフを図 2.4 に示す。各点は関係 R_i を表し、 TL_i は R_i の目標属性集合 (なければ ϕ) を、点 R_i と R_j 間の

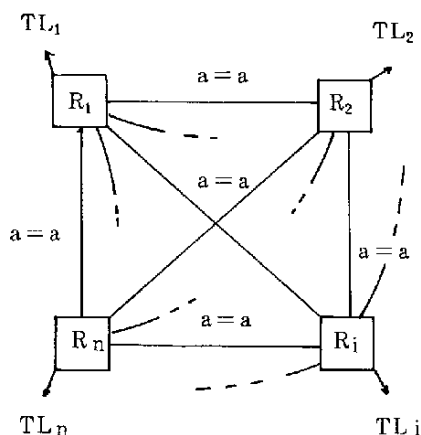


図 2.4 単純問合せ

辺は等結合述語 $R_i, a = R_j, a$ を表す。又関係 R_i は、 DM_i に存在するものとする ($D_{R_i} = DM_i$)。

あるデータモジュール DM_i が関係 R_i を放送網上に送出すると、他の全ての DM_j は R_i を同時に受信し、等結合演算 $R_j \leftarrow R_j [a = a] R_i$ を行える。各 DM_j は、結合処理の応答として、結合結果の関係 R_j のカーディナリティ、各属性のカーディナリティ等

を放送すると、全ての DM_k は DM_j の結合結果 R_j の大きさを知ることができる。よって、この次に、各 DM_i の有する関係 R_i の中で、最少のものを、全 DM_i が決定出来る。このことは、各 DM が完全分散制御と動的決定を行えることを意味している。以下に、単純問合せに対する処理アルゴリズム BS [TAKIM81] を示す。

[アルゴリズム BS]

(0) DM 間問合せ Q の参照する関係集合を $\Sigma = \{ R_1, \dots, R_n \}$ ($n \geq 2$) とし、 a を結合属性とし、任意の2つの関係 R_i と R_j ($i \neq j$) ($\in \Sigma$) 間に等結合 $R_i, a = R_j, a$ が存在するとする。 TL_i を R_i の目標属性集合とする。各 DM_i は、 Σ 内の全ての関係 R_j ($\in \Sigma$) について、 $R_j[a]$ のカーディナリティ ($R_j[a]$) の情報を有しているとする。*) $\Sigma' = \Sigma$ 。

(1) 各 DM_i は、 $\Sigma = \phi$ ならば (5)へ。 $\Sigma \neq \phi$ ならば、関係 $R_j \in \Sigma$ の中で $R_j[a]$ のカーディナリティ $card(R_j[a])$ が最少のものを見つける。これを R_j とする。各 DM_i は、 $\Sigma' \leftarrow \Sigma' - \{ R_j \}$ 。 $TL_{R_j} = \phi$ ならば、 $\Sigma \leftarrow \Sigma - \{ R_j \}$ ；

(2) DM_j は、 $R'_j \leftarrow R_j[a]$ ； R'_j を放送する。(6)へ。

(3) $\Sigma' \neq \phi$ でなければ、 Σ' 内の関係 R_i を持つ DM_i は R'_j を待つ。受信したならば、 $R_i \leftarrow R_i [a = a] R'_j$ ； $card(R_i[a])$ を求め応答 ACK_i (acknowledgement) に乗せて放送する。 Σ' 内の関係を持つ他の全ての DM_h からの応答 ACK_h を待つ。

(4) Σ' 内の関係を持つ DM_k は、 DM_h からの ACK_h を受けたならば、 $card(R_h[a])$ を取り出す。全ての ACK_h を受けたならば、(1)へ。

*) これは、各 DM_i が ILQP の結果関係のカーディナリティ情報を、ILQP の応答として放送することによって可能である。

(5) $\Sigma' = \phi$ であれば、 Σ 内の関係を持つ全 DM_i は、 R_j を待つ。受信したならば、 $R_i \leftarrow R_i$
 $[a=a] R_j'; R_i$ を結果 $DM(RDM)$ (利用者のいる DM) に送り、停止する。

RDM は、 Σ 内の全ての関係を待ち、受信したならば、関係を a について結合して、目標属性
 集合 TL で射影して、目標関係を得る。

(6) DM_j は、目標属性を持たない ($TL_{R_j} = \phi$) ならば、1 つでも ACK を受けたならば停止す
 る。 $TL_{R_j} \neq \phi$ ならば、(1)へ。□

アルゴリズム BS における通信コスト C_{BS} と処理コスト P_{BS} 、及び応答時間 R_{BS} について
 考える。まず、以下を定義する。

$\alpha_i \triangleq$ 結合属性 a の関係 R_i での選択度 [HEVNA78]

$\Sigma \triangleq \{R_1, \dots, R_n\}$

ここで、 Σ 内の関係は、 BS の(1)で選択された順に全順序づけられているとする。

$S_i \triangleq$ 関係 R_i の最初のサイズ

$C(x) \triangleq$ 量 x の通信コスト $\triangleq a + b \cdot x$

$P(S_i, S_j) \triangleq$ $|R_i| = S_i, |R_j| = S_j$ なる関係 R_i と R_j の結合 ($R_i[a=a]R_j$)
 コスト

$$\begin{aligned} C_{BS} &= C(S_1 + \alpha_1 S_2 + \alpha_1 \alpha_2 S_3 + \dots + \alpha_1 \alpha_2 \dots \alpha_{n-2} S_{n-1} + \alpha_1 \alpha_2 \dots \alpha_{n-1} S_n) \\ &= C(S_1 + \alpha_1 [S_2 + \alpha_2 [S_3 + \dots + \alpha_{n-2} [S_{n-1} + \alpha_{n-1} S_n] \dots]]) \\ &= S_1 + \alpha_1 [S_2 + \alpha_2 [S_3 + \dots + \alpha_{n-2} [S_{n-1} + \alpha_{n-1} S_n] \dots]] \quad \left(\begin{matrix} a=0 \\ b=1 \end{matrix} \right) \end{aligned}$$

第1項は、最初の関係 R_1 の放送コストである。第 i 項は、 i 番目の関係の放送コストである。

これは、 $R_1, \dots, R_{i-1}$ との結合によって制限されているので、量 $\alpha_1 \dots \alpha_{i-1} S_i$ を放送
 することになる。

$$\begin{aligned} P_{BS} &= \sum_{i=2}^n P(S_1, S_i) + \sum_{i=3}^n P(\alpha_1 S_2, \alpha_1 S_i) + \dots \\ &\quad + \sum_{i=n-1}^n P(\alpha_1 \dots \alpha_{n-3} S_{n-2}, \alpha_1 \dots \alpha_{n-3} S_{n-1}) + \\ &\quad + \sum_{i=n}^n P(\alpha_1 \dots \alpha_{n-2} S_{n-1}, \alpha_1 \dots \alpha_{n-2} S_n) \\ &= \sum_{j=1}^{n-1} \left(\sum_{i=j+1}^n P(\alpha_1 \dots \alpha_{j-1} S_j, \alpha_1 \dots \alpha_{j-1} S_i) \right) \end{aligned}$$

ここで、 $P(x, y) = x + y$ とすると

$$P_{BS} = \sum_{j=1}^{n-1} \left(\alpha_1 \cdot \alpha_2 \cdots \alpha_{j-1} \sum_{i=j+1}^n (S_j + S_i) \right)$$

$$R_{BS} = R_{CBS} + R_{PBS}$$

ここで、 $R_{CBS} = C_{BS}$

$$R_{PBS} = \sum_{j=1}^{n-1} \left(\max(\{P(\alpha_1 \cdots \alpha_{j-1} S_j, \alpha_1 \cdots \alpha_{j-1} S_i) \mid i = j+1, \dots, n\}) \right)$$

$P(x, y) = x + y$ とすると

$$\begin{aligned} R_{PBS} &= \sum_{j=1}^{n-1} \left(\max_i \left(\{ \alpha_1 \cdots \alpha_{j-1} (S_j + S_i) \mid i = j+1, \dots, n \} \right) \right) \\ &= \sum_{j=1}^{n-1} \left(\alpha_1 \cdots \alpha_{j-1} \left[(n-j) S_j + \max_i \left(\{ S_i \mid i = j+1, \dots, n \} \right) \right] \right) \end{aligned}$$

ここで、 $S_{j+1} \leq S_{j+2} \leq \dots \leq S_n$ なので

$$= \sum_{j=1}^{n-1} \left(\alpha_1 \cdots \alpha_{j-1} \left[(n-j) S_j + S_{j+1} \right] \right)$$

よって応答時間は、次の様になる

$$\begin{aligned} R_{BS} &= S_1 + \alpha_1 \{ S_2 + \alpha_2 \{ S_3 + \alpha_3 \{ S_4 + \dots + \alpha_{n-2} \{ S_{n-1} + \alpha_{n-1} S_n \} \} \} \} \\ &\quad + \sum_{j=1}^{n-1} \left(\alpha_1 \cdots \alpha_{j-1} \left[(n-j) S_j + S_{j+1} \right] \right) \end{aligned}$$

2.3.3 ビットマップ応答の利用 — アルゴリズムBBS

A. アルゴリズム

アルゴリズムBSでは、ある時点での関係 $R_j (= R_i[a])$ の放送に対して、各 DM_j は結合 $R_j \leftarrow R_j[a=d] R_i$ を行い、次に結合結果の中で $\text{card}(R_j[a])$ の最小の $R_j' (= R_j[a])$ を放送していく。ここで、 DM_j の放送する R_j は必ず R_i の部分集合であり ($R_j \subseteq R_i$)、各 DM_h は R_i を受信しているので、 $R_j' = R_j[a]$ を放送しなくても、 R_j' の R_i に対するビットマップ BM_j を送るだけで十分である。ビットマップ BM_j の大きさは、 R_j' に対してほとんどの場合十分小さい ($\text{card}(R_i')$ のビット数だけのメッセージでよい) ので、 R_i を受けた各 DM_j は、応答 ACK_j に、このビットマップ BM_j を乗せることが出来る。各 DM_k は他の DM_j からのビットマップ BM_j を受信したならば、積 $BM_k \leftarrow BM_k \wedge BM_j$ を行くと、全ての ACK を受けた時点では、同一のビットマップを得れることになる。よって、 DM 間問合せ Q を処理する為には、1つの関係 $R_i' (= R_i[a])$ の放送と、 $n-1$ 個の応答の放送によって処理出来ることになる。以下に、このアルゴリズムBBSを示す。

〔アルゴリズムBBS〕

(0)DM間問合せQの参照する関係集合を $\Sigma = \{ R_1, \dots, R_n \}$ 、 a を結合属性とする。 TL_i を関係 R_i の目標属性集合とする。各 DM_i は、他の DM_j 内の R_i の $card(R_j[a])$ を知っているとする。 $\Sigma' = \Sigma$ とする。

(1) $R_i \in \Sigma$ の中で、 $card(R_i[a])$ の最少の R_i を選ぶ。

(2) DM_i は、 $R_i' \leftarrow R_i[a]$ ； R_i' を放送する。 R_i' の R_i' に対するビットマップ BM_i （全てのビットはオン）をつくる。 $k \leftarrow i$ として(4)へ。

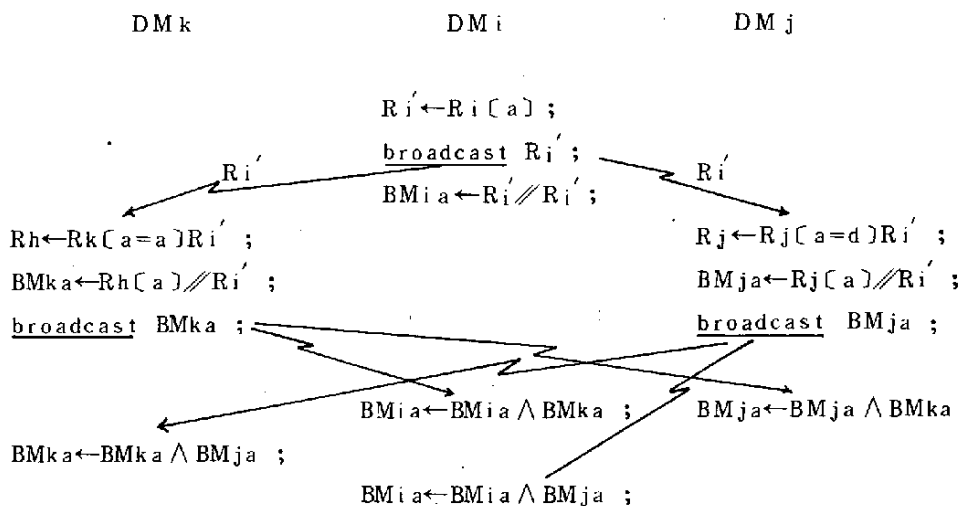
(3) Σ 内の関係 $R_j (\neq R_i)$ を持つ DM_j は、 R_i' を待つ。受信したならば、 $R_j \leftarrow R_j[a = a] R_i'$ ； $R_j[a]$ の R_i' に対するビットマップ BM_j をつくり、応答 ACK_j に乗せて放送する。

(4) $TL_{R_k} \neq \emptyset$ なる全ての DM_k は、他の DM_j からの応答 ACK_j を待つ。 ACK_j を受信したならば、 $BM_k \leftarrow BM_k \wedge BM_j$ ；全て受信したならば、 BM_k によって、 R_k を制限する。 R_k を結果 $DM(RDM)$ に送り、停止する。

結果 DM は、 $TL_{R_k} \neq \emptyset$ なる R_k を受信して、Qの目標関係 R をつくる。

(5) $TL_{R_k} = \emptyset$ ならば、停止する。□

図2.4は、アルゴリズムBBSにおける通信処理を図示している。



ここで、値集合 V 、 V の部分集合を V' とする。この時、演算 $V' // V$ は、 V に対する V' のbit-mapを生成する演算とする。

図2.4 アルゴリズム BBS

B. コ ス ト

次に、アルゴリズム BBS における全通信時間 C_{BBS} ，全処理時間 P_{BBS} ，及び応答時間 R_{BBS} について考える。まず、 n 個の関係集合を $\Sigma = \{R_1, \dots, R_n\}$ とし、 Σ 内の各関係 R_i と R_j 間に属性 a についての等結合 $R_j \cdot a = R_i \cdot a$ が存在する単純問合せを Q とする。各関係 $R_i \in \Sigma$ は、データモジュール DM_i に存在するとする。このとき、 Q がアルゴリズム BBS によって次の様に処理されたとする。

- (1) DM_j が $R'_1 (= R_1[a])$ を放送する。
 - (2) DM_i ($i \geq 2$) は、 R'_1 を受信して、結合演算 $R_i \leftarrow R_i[a=a] R'_1$ を行う。 $R_i[a]$ の R'_1 に対するビットマップ $BM_i (= R_i[a] \parallel R'_1)$ を設けて、 ACK_i に BM_i を乗せて放送する。
 - (3) 全データモジュール DM_j は、他の DM_i から ACK_i を受けたならば、 $BM_j \leftarrow BM_i \wedge BM_j$ 。全ての ACK を受信したならば、 BM_j をもとにして、 R_j を制約する。
- この通信処理スケジュールを S とする。この S の C_{BBS} ， P_{BBS} ， R_{BBS} を考える。まず、全通信時間 C_{BBS} は、次の様になる。

$$C_{BBS} \triangleq C(|R_1[a]|) + (n-1)C(\log_8 \text{card}(R_1[a])) \dots \dots (1)$$

第1項は、 $R'_1 (= R_1[a])$ を DM_1 から $DM_2, \dots, DM_2, \dots, DM_n$ に放送する通信時間である。第2項は、各 DM_i ($i = 2, 3, \dots, n$) が、ビットマップ BM_i を $DM_1, \dots, DM_{i-1}, DM_{i+1}, \dots, DM_n$ に放送する時間である。ビットマップ BM_i は、 R'_1 のカーディナリティ分のビットがあればよい。ここで、 $C(x) = x$ と仮定すると、(1)式は、(2)式の様になる。

$$C_{BBS} \triangleq |R_1[a]| + (n-1) \cdot \log_8 \text{card}(R_1[a]) \dots \dots (2)$$

各データモジュールでの処理時間の総和としての全処理時間 P_{BBS} は次の様である。

$$\begin{aligned} P_{BBS} \triangleq & Pr(R_1[a]) + \sum_{i=2}^n Pj(R_i[a=a]R'_1) \\ & + \sum_{i=1}^n Pb(R_i * BM_i) \\ & + \sum_{i=1}^n \sum_{j=2 \atop (i \neq j)}^n Pm(BM_i \wedge BM_j) \dots \dots (3) \end{aligned}$$

$Pr(R_1[a])$ は、 R_1 の a 上の射影 $R'_1 = R_1[a]$ を行う処理時間である。 $Pj(R_i[a=a]R'_1)$ は、結合 $R_i[a=a]R'_1$ を行う処理時間である。 $Pb(R_i * BM_i)$ は、ビットマップ BM_i を用いて R_i を制約する処理時間である。最後の項の $Pm(BM_i \wedge BM_j)$ はビット

マップの積 $BM_i \leftarrow BM_i \wedge BM_j$ を行う処理時間である。ここで、 $P_m(BM_i \wedge BM_j)$ は、各 DM_i の主記憶内で処理出来るとして無視する。第3項の $P_b(R_i * BM_i)$ は、ビットマップによる関係 R_i の制限コストである。第4項はメモリ内で処理出来るなるとして無視できる。ここで、次の仮定を設ける。

$$Pr(R_i[a]) = a \cdot \text{card}(R_i)$$

$$P_j(R_i[a=a]R_i') = b(\text{card}(R_i) + \text{card}(R_i'))$$

$$P_b(R_i * BM_i) = c \cdot \text{card}(R_i)$$

C. 障害回復

アルゴリズム BBS において、 R_i' の受信をある DM_j が失敗した時を考える。

ここで、 $S = \{DM_{j_1}, \dots, DM_{j_l}\} (l \geq 0)$ を、 R_i' を正しく受信出来た DM の集合とし、 $F = \{DM_{j_1}, \dots, DM_{j_m}\} (m \geq 0)$ を R_i' の受信に失敗した DM の集合とする。ここで、 $S \cup F = \{DM_i\}$ は、全 DM の集合 D となる。各 $DM_{i_h} \in S$ は、他の S 内の $DM_{i_h'}$ から bit-map $BM_{i_h'a}$ を受信し、bit-map の積演算まで行えるとする。各 $DM_{i_j} \in S$ は、この bit-map 演算後、各 $DM_{j_k} \in F$ からの bit-map $BM_{i_k'a}$ の着信を待つ。この bit-map が着信しない (time-out (t_o)) によって検出) 時には、 S 内のある DM_{i_h} が現在までの $BM_{i_h'a}$ を用いて、 R_i' を制限した関係を R_i' を求め、放送する。

F 内の DM_{j_k} は、 R_i' を受信出来たならば、結合処理を行い、bit-map を放送し、 S 内の DM_{i_h} と共に bit-map の処理に加わる。

F が ϕ になるまで上記をくり返す。

2.3.4 一般問合せ

一般問合せ Q とは、異ったデータモジュール上の関係 $R_1, \dots, R_n (n \geq 2)$ 間の結合属性として複数 $(a_1, \dots, a_m) (m \geq 2)$ がある問合せである。一般問合せの処理方法としては、次の2つがある。

(i) 制限方法

(ii) 同定方法

(i) の制限方法では、利用者のいる結果 $DM(RDM)$ に目標属性値を集める為に、なるべく各 DM の関係を DM 間通信処理によって制限してから送ろうとする。しかし、各関係 R_i と R_j 間の結合 $R_i.a_k = R_j.a_k$ に対して、 RDM に送る時、 $R_i[a_k] = R_j[a_k]$ とは限らない。例として [HEVNA78] がある。(ii) の同定方法では、全ての結合式 $R_i[a_k] = R_j[a_k]$ について、DM 間通信処理によって、 $R_i[a_k] = R_j[a_k]$ としてから、 RDM に R_i と R_j を転送する。

例として、図 2.5 を考える。

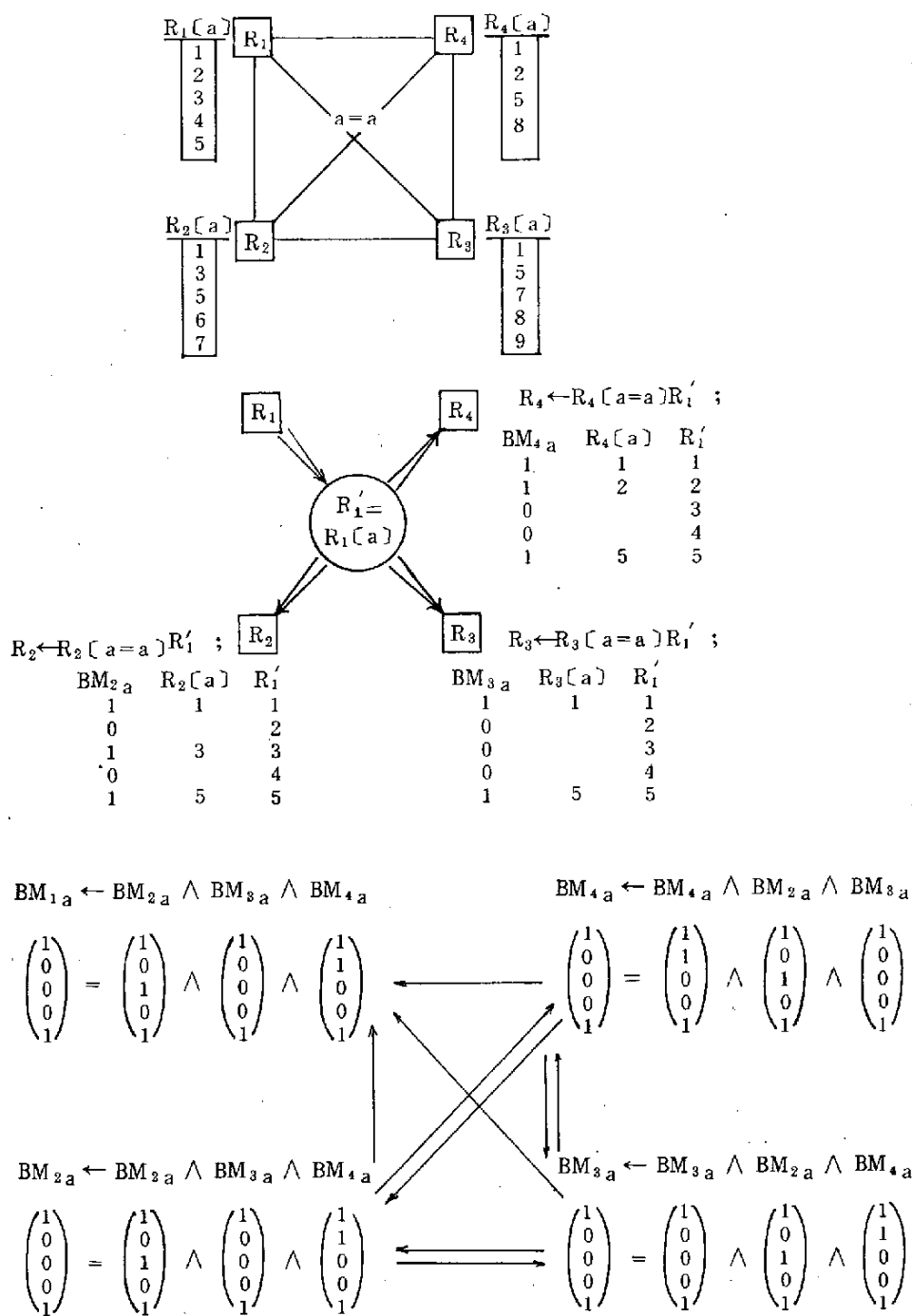


図 2.5 BBS の例

ここで、以下の記号を定義する。

$Q \triangleq$ 一般問合せ

$\Sigma \triangleq \{R_1, \dots, R_n\} \triangleq Q$ の参照する関係の集合

$TL \triangleq Q$ の目標属性集合

$A \triangleq Q$ 内の結合属性の集合

ここで、等結合によって互いに連結な全ての結合属性は、同一名となる様に再名前づけがなされているとする。

各関係 $R_i \in \Sigma$ に対して

$\Omega_{R_i} \triangleq R_i$ の有する属性の集合

$D_{R_i} \triangleq R_i$ の存在するデータモジュール

$\triangleq DM_i$

$A_{R_i} \triangleq R_i$ 内の結合属性集合

$TL_{R_i} \triangleq R_i$ 目標属性集合

ここで、 $\Omega_{R_i} = A_{R_i} \cup TL_{R_i}$ である。

各結合属性 $a \in A$ に対して、

$R_a \triangleq$ 結合属性 a を持つ関係の集合

$\triangleq \{R \mid a \in A_R\}$

例えば、図 2.6 は一般問合せである。

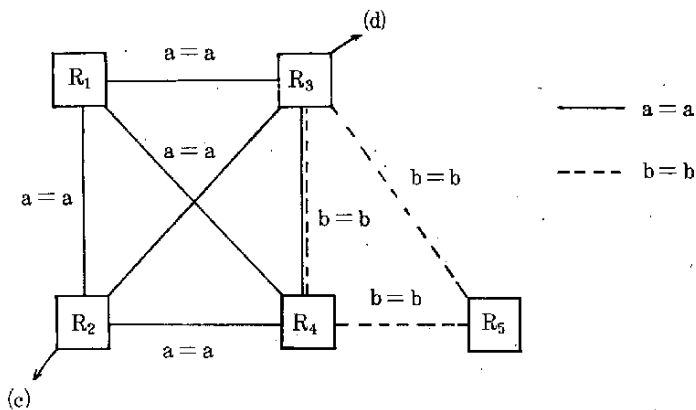


図 2.6 一般問合せの例

図 2.6 では

$$\begin{aligned}
 \Sigma &= \{ R_1, R_2, R_3, R_4, R_5 \} \\
 A &= \{ a, b \} \\
 TL &= \{ c, d \} \\
 \Omega_{R_1} &= \{ a \} & AR_1 &= \{ a \} & TLR_1 &= \phi \\
 \Omega_{R_2} &= \{ a, c \} & AR_2 &= \{ a \} & TLR_2 &= \{ c \} \\
 \Omega_{R_3} &= \{ a, b, d \} & AR_3 &= \{ a, b \} & TLR_3 &= \{ d \} \\
 \Omega_{R_4} &= \{ a, b \} & AR_4 &= \{ a, b \} & TLR_4 &= \phi \\
 \Omega_{R_5} &= \{ b \} & AR_5 &= \{ b \} & TLR_5 &= \phi
 \end{aligned}$$

2.3.5 制限方法による一般問合せ処理

A. アルゴリズム

まず制限方法について考える。制限方法としては、2.3.3 で述べたアルゴリズム BBS を基本にして考える。各結合属性 a_i ($i=1, \dots, m$) 毎に、アルゴリズム BBS の手順を行って、各データモジュール (DM) 上の関係を制限した後、全ての関係を結果 DM (RDM) に送り、RDM で結合属性についての結合を行い、目標関係を得る方法である。このアルゴリズムを GBS とする。

[アルゴリズム GBS]

(0) 各データモジュール (DM) は、他の DM 内の関係 R の結合属性 a の射影 $R[a]$ のサイズ (又は、カーディナリティ) についての情報を有しているとする。

$$A' \leftarrow A ; \Sigma' \leftarrow \Sigma ; \pi \leftarrow \phi ;$$

各 R について $\pi_R \leftarrow \phi ; A'_R \leftarrow AR ;$

(1) 各 DM は、各結合属性 $a \in A'$ と関係 $R \in Ra$ の中で、 $\text{card}(R[a])$ の最少のものを見つける。これを R_i と a とする。

$$\text{card}(R_i[a]) = \min_{\substack{\forall a \in A \\ \forall R \in Ra}} (\text{card}(R[a]))$$

ここで、 R_i と a を各々、ソース関係と、ステージ属性と呼ぶ。

$$\pi \leftarrow \pi \cup \{a\} ; A' \leftarrow A' - \{a\} ;$$

(2) R_i を持つ DM i は、 $R'_i (= R_i[a])$ を放送する。

$$R'_i \leftarrow R_i[a] ;$$

$$\text{broadcast}(R'_i) ;$$

全ての DM j ($a \in AR_j$) からの応答 ACK $_j$ を待つ。(4)へ

(3) $a \in AR_j$ なる関係 R_j を持つ DM j は、 R'_i を待つ。 R'_i を受信したならば、

$\forall b \in \pi_{R_j}$ に対して、 $R_{j_b} \leftarrow R_i [b]$;

$R_j \leftarrow R_j [a=a] R'_i$; $A'_{R_j} \leftarrow A_{R_j} - \{a\}$;

結合属性 a について、ビットマップ BM_{ja} をつくる。

$BM_{ja} \leftarrow R_j [a] // R'_i$;

今までのステージ属性 $b \in \pi_{R_j}$ に対して、ビットマップ BM_{jb} をつくる。

$\forall b \in \pi_{R_j}$

if $R_i [b] \subset R_{ib}$ then $BM_{jb} \leftarrow R_j [b] // R_{jb}$;

DM_i は、応答 ACK_j に、 BM_{ja} と $\{BM_{jb}\}$ 、及び、結合属性 $c \in A'_{R_j}$ の $R_i [c]$ のカーディナリティを乗せて放送する。

$\pi_{R_j} \leftarrow \pi_{R_j} \cup \{a\}$;

他の全ての応答を待つ。

(4) DM_k が、 DM_j からの応答 ACK_j を受信したならば、

if $a \in A_{R_k}$ then $BM_{ka} \leftarrow BM_{ka} \wedge BM_{ja}$;

if $b \in \pi_{R_k} \cap \pi_{R_j}$ then $BM_{kb} \leftarrow BM_{kb} \wedge BM_{jb}$;

全ての応答を受けたならば、 BM_{ka} 、 $\{BM_{kb}\}$ をもとにして、 R_h を制限する。

(5) [$A' \neq \emptyset$ のとき] ある R_l が、 $TL_{R_l} = \emptyset$ (目標属性を持たない) で、かつ $A_{R_l} = \{a\}$ のとき、 Σ から R_l を除く ($\Sigma \leftarrow \Sigma - \{R_l\}$)。この結果、結合属性 a を持つ関係が、ある $R_h \in \Sigma$ だけとなり $a \in TL_{R_h}$ の時、 $\Omega_h \leftarrow \Omega_h - \{a\}$ (R_h から属性 a を除く);

$A \leftarrow A - \{a\}$;

(6) [$A' = \emptyset$ のとき] 全ての関係 $R \in \Sigma$ を、結果 $DM(RDM)$ に転送して、 RDM は、各結合属性 $a \leftarrow A$ について結合を取り、目標関係を生成する。□

2.3.6 同定方法による一般問合せ処理

次に、各データモジュール (DM) 内の関係を、問合せの条件を満たすものに限定した後に、利用者のいる結果 $DM(RDM)$ にこれらの関係を集めて結合を行う同定方法について考える。 $JDDBS$ では、本項で述べる同定方法としてのデータモジュール (DM) 間問合せ処理 (DMQ) アルゴリズムに基づいて通信処理が行われている。 DMQ アルゴリズムは、次の特徴を有している。

- (1) 関係及び制御情報の放送
- (2) 結合属性集合値のコード化
- (3) 通信処理スケジュールの動的決定
- (4) 完全分散制御

A. コード化

ある結合属性 $a \in A$ に対して、アルゴリズム BBS による処理を考える。即ち、

- (1) DM_i による $R_i (= R'_i [a])$ の放送

(2) 各 $DM_j (j \neq i)$ による結合とビットマップ生成

$$R_j \leftarrow R_j [a=a] R'_i;$$

$$BM_j \leftarrow R_j [a] // R'_i;$$

BM_j の放送

(3) 各 DM_k による全ての BM_j の受信と BM_k による R_h の制限

$$BM_k \leftarrow BM_k \wedge BM_j;$$

$$R_k \leftarrow R_k * BM_k;$$

この処理の終了時点で、全てのデータモジュール DM_i 内の関係 R_i の結合属性 a についての射影 $R_i [a]$ は全て等しくなる。即ち、

$\forall R_{di} \in Ra$ に対して

$$R_{di} [a] = RA_a.$$

結合属性 a が、目標属性でないとき ($a \notin TL$)、(1)~(3)の処理が終了した時点で、 $R_i [a]$ を表す為に、定義域 $dom(a)$ 内の値ではなく、あるコード化された値でも表せる。

我々は、次の方法によって、各 $R_i [a]$ をコード化して表す。

[コード化]

(1) 値集合 RA_a を、 a の値で昇順に整列させる。

(2) 各値 $v \in R_i [a] (= RA_a)$ を、整列された RA_a 内の順番によってコード化する。□

例えば、 $RA_a = \{ "A", "DE", "B", "F", "AB" \}$ は、 $\{ 0, 3, 2, 4, 1 \}$ として表す。ここで、

$\tilde{a} \triangleq (1), (2)$ によって結合属性 a がコード化された結合属性

$$dom(\tilde{a}) \triangleq \{ 0, 1, 2, \dots, card(RA_a) - 1 \}$$

$$|\tilde{a}| \triangleq a \text{ のサイズ}$$

$$\triangleq \lceil \log_2 (card(RA_a)) \rceil$$

$\tilde{a}$ を、 a のコード化 (された) 結合属性と呼ぶ。各関係 R 内のコード化結合属性の集合を $\tilde{Q}_R$ とする。

a が目標属性でもある場合には、属性 a の他に、コード化結合属性 $\tilde{a}$ を R に加える。

次に、ある結合属性 $b \in A (b \neq a)$ について、アルゴリズム BBS による処理がなされたとする。ここで、データモジュール DM_{b_1} が、 $R_{b'_1} = R_{b_1} [b]$ を放送し、

各 $DM_{b_i} (R_{b_i} \in R_b)$ で結合

$$(1) R_{b_i} \leftarrow R_{b_i} [b=b] R'_{b_1};$$

がなされたとする。関係 R_{b_i} は、既にコード化された属性集合 $\tilde{Q}_{R_{b_i}}$ を持つとすると、

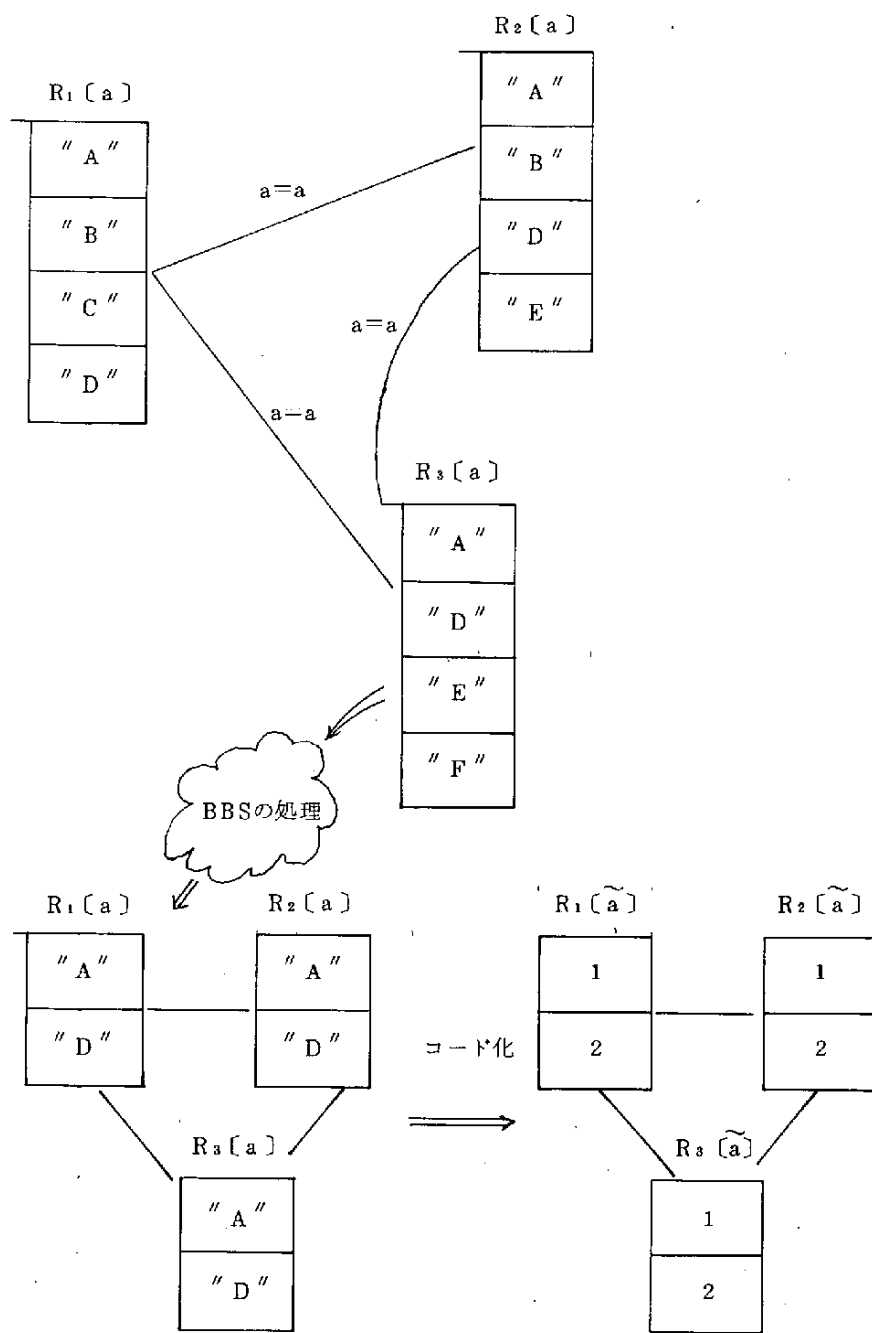


図 2.7 コード化の例

(1)の結合前の射影 $Rb_i[\tilde{c}]$ ($\tilde{c} \in \tilde{\Omega}_{Rb_i}$)は、結合(1)によって制約される。この為、ビットマップ $BMb_i (=Rb_i[\tilde{b}] \setminus Rb_i)$ に加えて、ビットマップ BM_{c_i} も放送する。この時、 BM_{c_i} では、 $Rb_i[\tilde{c}]$ 内に存在するコード(整数)に対応したビットがon (=1)で、他がoff (=0)となる。

B. 複数結合属性のコード化

一般問合せでは、複数の結合属性が存在する。各結合属性 $a \in A$ について、アルゴリズムGBSは、各関係 R の a についての射影 $R[a]$ を全て等しくできる。しかし、各関係の複数の結合属性の値の組を等しくできない。よって、これを行う為には、複数の結合属性値の組を、通信網を介して転送することは、通信オーバーヘッドを増大させることになる。この為、我々は、次の方法を用いる。

- (i) 結合属性値を、そのまま転送せずに、コード化された結合属性値を転送する。
 - (ii) コード化された結合属性値の組を更にコード化して、データを圧縮する。
 - (iii) ある結合属性集合 $\tilde{\Omega}$ について、アルゴリズムGBSで通信処理を行った後、各関係 R の $R[\tilde{\Omega}]$ は全て等しくなる。この為、 $R[\tilde{\Omega}]$ は全て等しくなる。この為、 $R[\tilde{\Omega}]$ を、コード化して1つのコード化属性 $\tilde{a}$ としてしまう。
- (i)、(ii)、(iii)は、次の様にして処理される。

(1) 関係 R_i と、コード化属性集合 $\tilde{\Omega}_{R_i}$ を考える。 DM_i は、 $\tilde{R}_i (=R_i[\tilde{\Omega}_{R_i}])$ を放送する。

(2) $\Omega_{R_i} \cap \Omega_{R_j} = \Omega_{ij} \neq \emptyset$ なる DM_j は、 $\tilde{R}_i$ を受信して、 $\tilde{\Omega}_{ij}$ について結合を行う。

$$R_j \leftarrow R_j[\tilde{\Omega}_{ij} = \tilde{\Omega}_{ij}] \tilde{R}_i;$$

この結果、 Ω_{R_j} は、 Ω_{R_j} と $\tilde{\Omega}_{R_i}$ の和となる。

$$\Omega_{R_j} \leftarrow \Omega_{R_j} \cup \tilde{\Omega}_{R_i};$$

$\tilde{\Omega}_{ij}$ についての結合の応答として、ビットマップ BM_{ji} を放送する。

$$BM_{ji} \leftarrow R_j[\tilde{\Omega}_{ij}] \setminus \tilde{R}_i;$$

(3) 全てのデータモジュール DM_k は、 BM_{ji} を受信したならば、

$$BM_{ki} \leftarrow BM_{ki} \wedge BM_{ji};$$

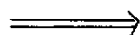
全ての応答を受信したならば、 R_k を BM_{ki} で制限する。

$R_k[\tilde{\Omega}_{ij}]$ を、昇順に整列させて、 $R_k[\tilde{\Omega}_{ij}]$ の各値を、この順番によってコード化する。 $\tilde{\Omega}_{ij}$ をコード化した属性を新たに $\tilde{C}_{ij}$ とする。

例えば、図2.8の様になる。

$R(\tilde{a}, \tilde{b}, t)$

1	2	"A"
3	6	"B"
5	3	"A"
8	1	"B"
10	4	"B"

 $(\tilde{a}, \tilde{b})$ のコード化

 $R(\tilde{c}, t)$

0	"A"
1	"B"
2	"A"
3	"B"
4	"B"

図 2.8 属性集合のコード化

C. 通信データのコード化

B. で、 $R_i[\tilde{\Omega}_{R_i}]$ が放送されるとき、通信量を、次の様にして、減少化させる。ここで、

$$\begin{aligned}\tilde{\Omega}_{R_i} &\triangleq \{\tilde{c}_1, \tilde{c}_2, \dots, \tilde{c}_l\} \quad (l \geq 2) \\ \text{dom}(\tilde{c}_i) &\triangleq \{0, 1, \dots, n_i - 1\} \quad (n_i \geq 1) \\ &\triangleq \text{コード化結合属性 } \tilde{c}_i \text{ の定義域}\end{aligned}$$

この時、各組 $r_i \in R_i[\tilde{\Omega}_{R_i}]$ を、次の様にしてコード化する。

$[R_i[\tilde{\Omega}_{R_i}]]$ のコード化

$\forall R_i = (v_1, \dots, v_l) \in R_i[\tilde{\Omega}_{R_i}]$ (ここで、 $\tilde{c}_i(r_i) = v_i$ ($i=1, \dots, l$)) に対して、

$$\begin{aligned}\tilde{r}_i &= v_1 + n_1 [v_2 + n_1 [v_2 + \dots + n_{l-1} [v_l] \dots]] \\ \tilde{r}_i &\text{を } r_i \text{ のコード化された組とする。} \square\end{aligned}$$

D. アルゴリズム

一般問合せ Q を、同定方法によって処理する為のアルゴリズム DMQ を、以下に示す。

$[\text{アルゴリズム } DMQ]$

(0) 全ての $R_i \in \Sigma$ に対して、

$$\tilde{\Omega}_{R_i} \leftarrow \phi; \quad A' \leftarrow A; \quad \tilde{\Omega} \leftarrow \phi;$$

(1) [終了条件のチェック]

$$\text{if } \text{check} = \text{true}, \text{ then go to} \quad (10)$$

(2) [ソース関係の決定]

各DMは、自分の有するグローバル情報 (GI) に基づいて

(i) $\forall a \in A' \quad \forall Ri \in \Sigma (a \in A_{Ri})$ に対して、 $|Ri[a]| = Ca$ が最少のものを選ぶ。

(ii) $\forall Ri \in \Sigma (\tilde{\Omega}_{Ri} \neq \phi \wedge \text{card}(\tilde{\Omega}_{Ri}) \geq 2)$ に対して、 $|Ri[\tilde{\Omega}_{Ri}]| = \tilde{C}_i$ が最少のものを選ぶ。

(iii) ここで、 $\tilde{C}_i < Ca$ ならば、(6)へ。

(3) [$Ri[a]$ の放送]

DMiは、 $R'i (= Ri[a])$ を放送し、(5)へ。

(4) a を結合属性としてもつ関係 $Rj (a \in A_{Rj})$ を有するDMjは、 $R'i$ を待つ。 $R'i$ を受信したならば、 Rj と $R'i$ の結合を行いビットマップ $BMja$ を生成する。

$$Rj \leftarrow Rj[a=a]R'i;$$

$$BMja \leftarrow Rj[a] \parallel R'i;$$

$\forall \tilde{c} \in \tilde{\Omega}_{Rj}$ に対して、

$$BMj\tilde{c} \leftarrow Rj[\tilde{c}] \text{ 内の値のビット位置をONとしたビットマップ};$$

$\forall b \in A_{Rj} - \{a\}$ に対して、

$$Cjb \leftarrow \text{card}(Rj[b]);$$

結合の応答として、ビットマップ $BMja$, $\{BMj\tilde{c}\}$, $\{Cjb\}$ を乗せて $ACKj$ を放送する。

(5) 全てのDMkは、全ての応答 $ACKj$ を待つ。 $ACKj$ を受信したならば、

$$BMka \leftarrow BMka \wedge BMja;$$

$\forall \tilde{c} \in \Omega_{Rk} \cap \tilde{\Omega}_{Rj}$ に対して、

$$BMk\tilde{c} \leftarrow BMk\tilde{c} \wedge BMj\tilde{c};$$

$ACKj$ の転送してきた、未結合属性のカーディナリティ情報を、DMj内のグローバル情報 (GI) にセットする。

全ての応答を受信したならば、 $BMja$, $\{BMj\tilde{c}\}$ を用いて、 Rk を制限する。

$Rk[a]$ をコード化する。

$$A_{Rk} \leftarrow A_{Rh} - \{a\}; \quad \tilde{\Omega}_{Rh} \leftarrow \tilde{\Omega}_{Rh} \cup \{\tilde{a}\};$$

$$A' \leftarrow A' - \{a\};$$

$$\tilde{\Omega} \leftarrow \tilde{\Omega} \cup \{\tilde{a}\};$$

(2) へ。

(6) [$Ri[\tilde{\Omega}_{Ri}]$ の放送]

DMiは、 $\tilde{R'i} (= Ri[\tilde{\Omega}_{Ri}])$ を放送する。(8)へ。

(7) $\tilde{\Omega}_{Ri}$ 内のコード化属性を持つ関係 $Rj (\tilde{\Omega}_{Ri} \cap \tilde{\Omega}_{Rj} = \tilde{\Omega}_{ij} \neq \phi)$ を持つ全ての

DMjは、 $\tilde{R'i}$ を待つ。 $\tilde{R'i}$ を受信したならば、

$$R_j \leftarrow R_j \setminus \tilde{\Omega}_{ij} = \tilde{\Omega}_{ij} \setminus \tilde{R}_i;$$

$$BM_j \leftarrow R_j \setminus \tilde{\Omega}_{ij} \setminus \tilde{R}_i;$$

全ての $\tilde{c} \in \tilde{\Omega}_{R_j} = (\tilde{\Omega}_{R_j} - \tilde{\Omega}_{R_i})$ ($\tilde{\Omega}_{R_i}$ 外のコード化属性) に対して、

$$BM_j \tilde{c} \leftarrow R_j \setminus \tilde{c} \text{ の値のビット位置を } on (=1) \text{ としたビットマップ};$$

$\forall b \in A_{R_j}$ に対して、

$$C_{jb} \leftarrow \text{card}(\{R_j \setminus a\});$$

結合処理の応答として、 BM_j , $\{BM_j \tilde{c}\}$, $\{C_{jb}\}$ を ACK_j に乗せて放送する。

$$\tilde{\Omega}_{R_j} \leftarrow \tilde{\Omega}_{R_j} \cup \tilde{\Omega}_{R_i};$$

(8) 各 DM_k は、全ての DM_j ($k \neq j$) からの応答 ACK_j を待つ。 ACK_j を受信したならば、

$$\tilde{\Omega}_{R_i} \subseteq \tilde{\Omega}_{R_k} \text{ の時}$$

$$BM_k \leftarrow BM_k \cap BM_j;$$

$\forall \tilde{c} \in \tilde{\Omega}_{R_k} \cap \tilde{\Omega}_{R_j}$ に対して、

$$BM_k \tilde{c} \leftarrow BM_k \tilde{c} \cap BM_j \tilde{c};$$

ACK_j 内の C_{jb} を、 DM_k のグローバル情報にセットする。

全ての応答を受信したならば、ビットマップを用いて、 R_k を制約する。 $R_k \setminus \tilde{\Omega}_{R_i}$ をコードして $R_k \setminus \tilde{c}_i$ とする。

$$\tilde{\Omega}_{R_k} \leftarrow \tilde{\Omega}_{R_k} - \{\tilde{\Omega}_{R_k}\} \cup \{\tilde{c}_i\};$$

$$\tilde{\Omega} \leftarrow \tilde{\Omega} - \{\tilde{\Omega}_{R_i}\} \cup \{\tilde{c}_i\} \quad (2) \sim.$$

(9) [結果 $DM(RDM)$ に、関係を転送]

$TL_{R_i} \neq \phi$ なる R_i を有する DM_i は、 R_i を RDM に転送する。 RDM は、送られてきた関係 R_i を $\tilde{\Omega}$ で結合する。

$$R \leftarrow \phi;$$

for $\forall R_i \in \Sigma(TL_{R_i} \neq \phi)$

$$\text{do } R \leftarrow R \setminus (\tilde{\Omega} - \tilde{\Omega}_{R_i}) R_i;$$

$$R \leftarrow R \setminus \bigcup_{R_i} TL_{R_i};$$

R を利用者に出力する。□

アルゴリズム DMQ では、各 DM がグローバル情報 (GI) を用いてまず、コード化されていない属性の中で、サイズ $|R_i \setminus a| (=Ca)$ を最少とする R_i と a を見つける。同時に、コード化された結合属性の中で、サイズ $|R_i \setminus \tilde{\Omega}_{R_i}| (= \tilde{c}_i)$ を最少とする R_i を見つける。ここで、 $\tilde{c}_i$ が Ca より小さいならば、 $\tilde{\Omega}_{R_i}$ についての結合処理を行う ((6), (7), (8))。逆であれば、 a についての GBS 処理を行う ((3), (4), (5))。

(3), (4), (5) は、2, 3, 5 のアルゴリズム GBS による処理である。

(6), (7), (8) では、既に GBS 処理によって、コード化された結合属性集合についての処理がなされる。各関係 R_i のコード化結合属性集合 $\tilde{\Omega}_{R_i}$ が、2 つ以上のコード化結合を持つ

ているとき、この中で、 $|R_i[\tilde{\Omega}_{R_i}]|$ の最少の $\tilde{R}'_i (=R_i[\tilde{\Omega}_{R_i}])$ が、放送される(6)。

$\tilde{\Omega}_{R_i}$ 内のコード化結合属性を、ある関係 R_j が持つとき(即ち、 $\tilde{\Omega}_{R_i} \cap \tilde{\Omega}_{R_j} = \tilde{\Omega}_{ij} \neq \emptyset$)、 $\tilde{\Omega}_{R_i}$ と $\tilde{\Omega}_{R_j}$ の共通コード化結合属性集合を $\tilde{\Omega}_{ij}$ とする。 DM_j は、 $\tilde{R}'_i$ を受信して、 R_j と $\tilde{\Omega}_{ij}$ について等結合を行う($R_j \leftarrow R_j[\tilde{\Omega}_{ij} = \tilde{\Omega}_{ij}]\tilde{R}'_i$)。この結果 R_j は、結合前の属性集合 Ω_{R_j} に加えて、 $\tilde{\Omega}_{R_i}$ を持つことになる。結合後、 $\tilde{R}'_i$ に対する $\tilde{R}_j[\tilde{\Omega}_{R_i}]$ のビットマップ BM_j をつくる。更に、 $\tilde{\Omega}_{R_i}$ に含まれない R_j のコード化属性 $\tilde{c}$ について、ビットマップ $BM_j\tilde{c}$ をつくる。コード化されていない結合属性 b についての射影 $R_j[b]$ のカーディナリティ C_{jb} をもとめる。これらを、結合処理の応答 ACK_j に乗せて放送する(7)。

ACK_j を受信した DM_k は、ビットマップの積演算を行う。全ての ACK を受信したならばビットマップを用いて、 R_k を制約する。次に、 $R_k[\tilde{\Omega}_{R_i}]$ を整列させて、各値を整列順番でコードし、 $\tilde{\Omega}_{R_i}$ を1つのコード化属性 $\tilde{C}_i$ とする(8)。

終了条件 $check[E]$ (を参照)を満たした時には、各関係間の結合属性は、コード化された1つの属性 $\tilde{c}$ だけとなっている。よって、各 DM_i は、目標項目を有する($TL_{R_i} \neq \emptyset$)関係 R_i を、利用者のいる結果 $DM(RDM)$ に転送する。 RDM は、 $\tilde{c}$ について、これらの関係を結合し、目標関係を生成し、利用者に出力する。

アルゴリズム DMQ では、結合属性値をコード化し、かつ、複数の結合属性を1つのコード化属性に縮退されていく。これによって、通信量を減少するとともに、処理コストと格納コストを減少できる。

(3), (4), (5)を例で示す。図2.9は、3つの関係 R_1, R_2, R_3 の間に $a=a$ の結合がある場合である。(3)で、 $R_1[a](=R'_1)$ が放送され、 DM_2 と DM_3 で各々、結合 $R_2 \leftarrow R_2[a=a]R'_1$; $R_3 \leftarrow R_3[a=a]R'_1$; がなされたとする。各々のビットマップ $BM_{1a}, BM_{2a}, BM_{3a}$ は図に示す通りになる。(5)で、ビットマップの積を取ると、図2.10の様に、 $R_1[a], R_2[a], R_3[a]$ は値として $\{\alpha, \gamma\}$ だけとなる。これをコード化すると、 $\{1, 2\} = \text{dom}(\tilde{a})$ となる。

図2.11, 2.12, 2.13は、アルゴリズムの(6), (7), (8)の例を示している。図2.11は、 $(R_1, \tilde{a}=R_3, \tilde{a} \text{ and } R_1, \tilde{b}=R_2, \tilde{b})$ を示している。属性 $\tilde{a}$ と $\tilde{b}$ は、既にコード化されているとする。 $R_1[\tilde{a}, \tilde{b}](=R_1)(A_{R_1} = \{\tilde{a}, \tilde{b}\})$ を放送すると、 $R_2 \leftarrow R_2[\tilde{b}=\tilde{b}]\tilde{R}_1$; $R_3 \leftarrow R_3[\tilde{a}=\tilde{a}]\tilde{R}_1$ が行われ、 $R_2(\tilde{a}, \tilde{b},), R_3(\tilde{a}, \tilde{b}, d)$ となる(図2.12)。 $R_2[\tilde{A}_{R_1}]$ と $R_3[\tilde{A}_{R_1}]$ の $\tilde{R}_1$ に対するビットマップ $\tilde{B}M_2, \tilde{B}M_3$ を放送し、ビットマップの積を取ると図2.13のようになる。 $R_i[\tilde{a}, \tilde{b}]$ は、コード化され、 $R_i[\tilde{a}\tilde{b}]$ となる。

ここで示してDQPアルゴリズムは、(2)の段階で、現在、転送量の最も小さいものを選び、かつ各 DM が独立して選べることから、動的な通信処理スケジュール決定と共に、完全分散

制御を実現している。

E. 終了条件

アルゴリズムDMQの終了条件 (check) (1) について考える。アルゴリズムDMQの終了条件としては、下記のものがある。

[終了条件]

- (1) 全ての結合属性がコード化されていて、

$$A' = \phi$$

- (2) 全ての関係 R_i が同一のコード化属性 $\tilde{\Omega}_{R_i} (= \tilde{\Omega})$ をもつ。□

F. 不要関係と属性の消去

GBSの実行中で、問合せの処理継続上、不要となった関係と属性を除去してしまうことは、問合せの処理の効率上有効である。全てのビットマップを受信した時点 (5) と (8) で、以下の関係と属性を除いてしまう。

[不要関係属性の除去]

$\forall R_i \in \Sigma$ に対して、

- (1) $TL_{R_i} = \phi$ (目標属性を R_i は持たない) で、かつ全ての結合属性がコード化されている ($A_{R_i} = \phi$, $\tilde{\Omega}_{R_i} = \{\tilde{c}\}$) とき、 R_i は今後の処理に不要であるので除く。

$$\Sigma \leftarrow \Sigma - \{R_i\};$$

- (2) 全ての結合属性がコード化され ($\tilde{\Omega}_{R_i} = \{\tilde{c}\}$) ているが $TL_{R_i} \neq \phi$ である。しかし、全ての目標属性 ($a \in TL_{R_i}$) を、他のある $R_k \in \Sigma$ が有している。この時も、 R_i は不要であるので除く。 $\Sigma \leftarrow \Sigma - \{R_i\};$

- (3) (1), (2)の結果、コード化属性 $\tilde{c} \in \tilde{\Omega}_{R_i}$ をどの関係ももたなくなった ($R_{\tilde{c}} = \{R_i\}$) 時、 $\tilde{c}$ を除く。

$$\tilde{\Omega}_{R_i} \leftarrow \tilde{\Omega}_{R_i} - \{\tilde{c}\};$$

$$\tilde{\Omega} \leftarrow \tilde{\Omega} - \{\tilde{c}\}; \quad \square$$

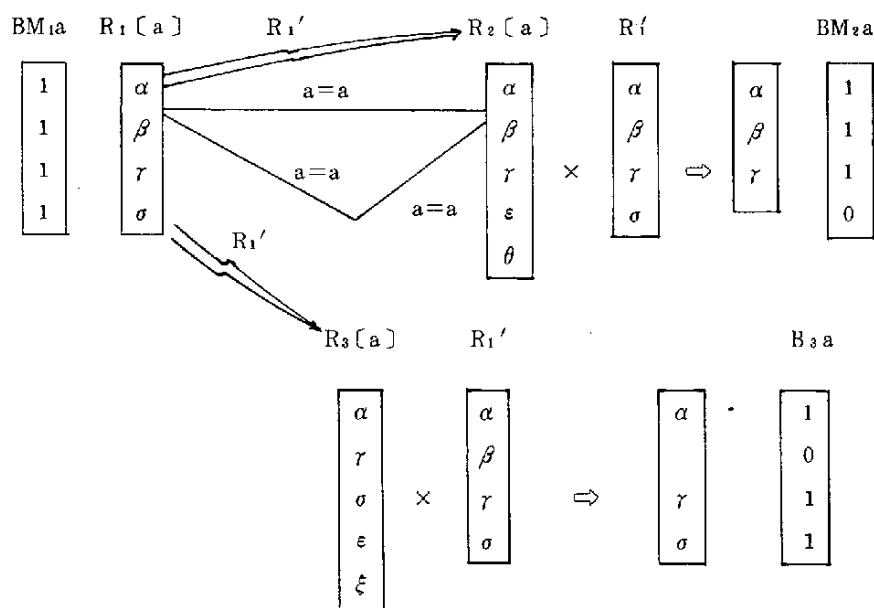


図 2.9 $R_1[a] (= R')$ の放送

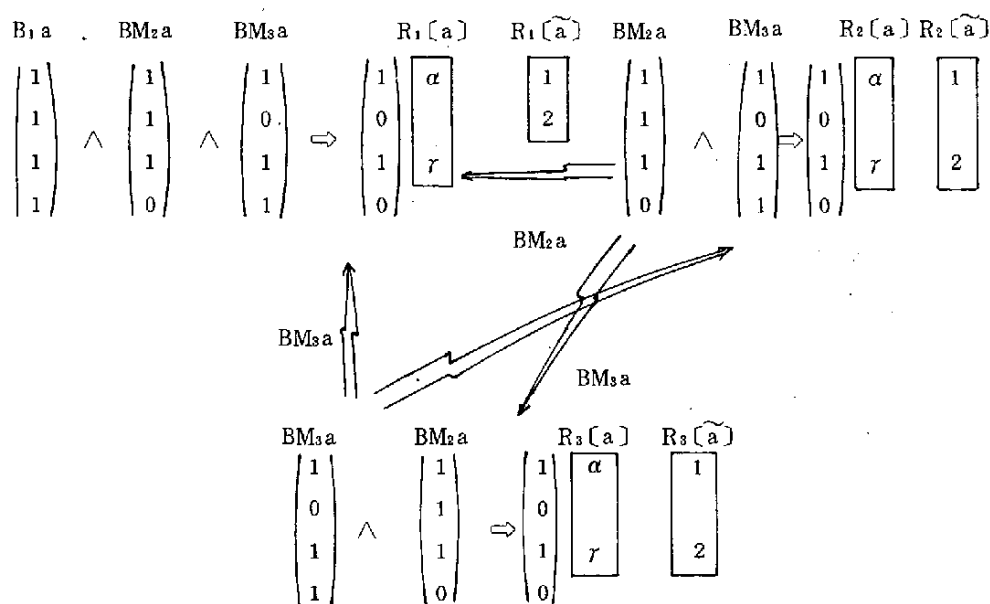


図 2.10 ビットマップ BM の放送

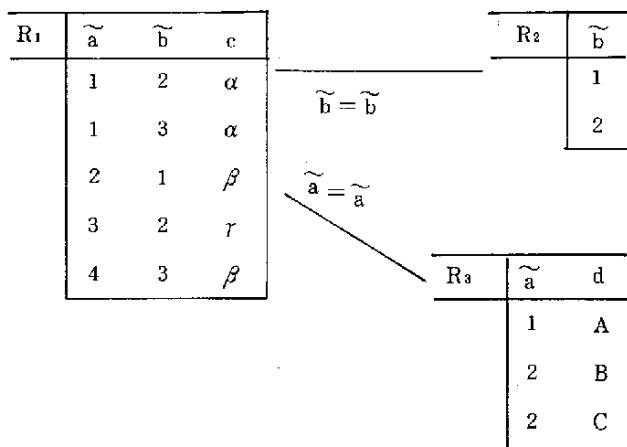


図 2.1 1 結 合

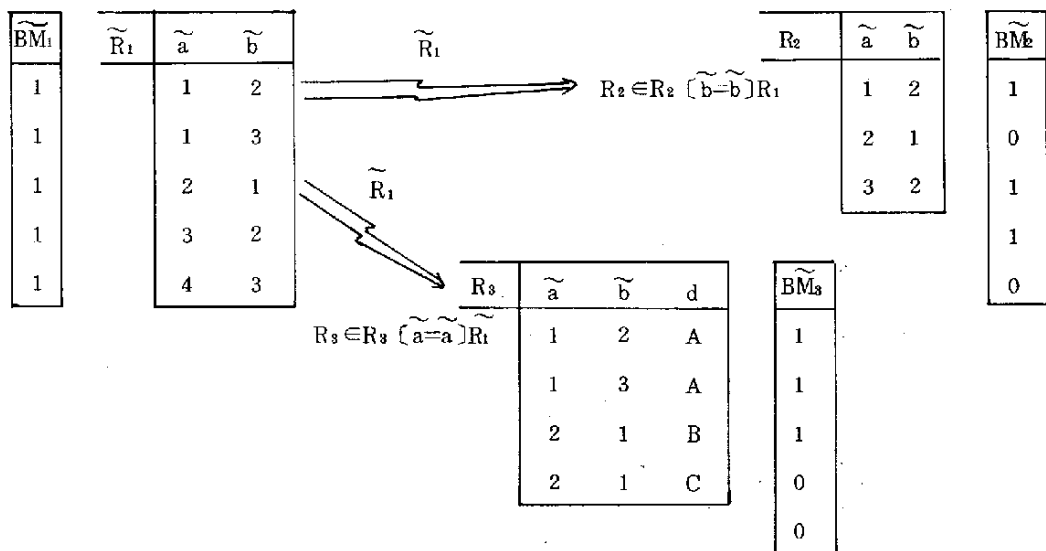


図 2.1 2 $\tilde{R}_1$ の 転 送

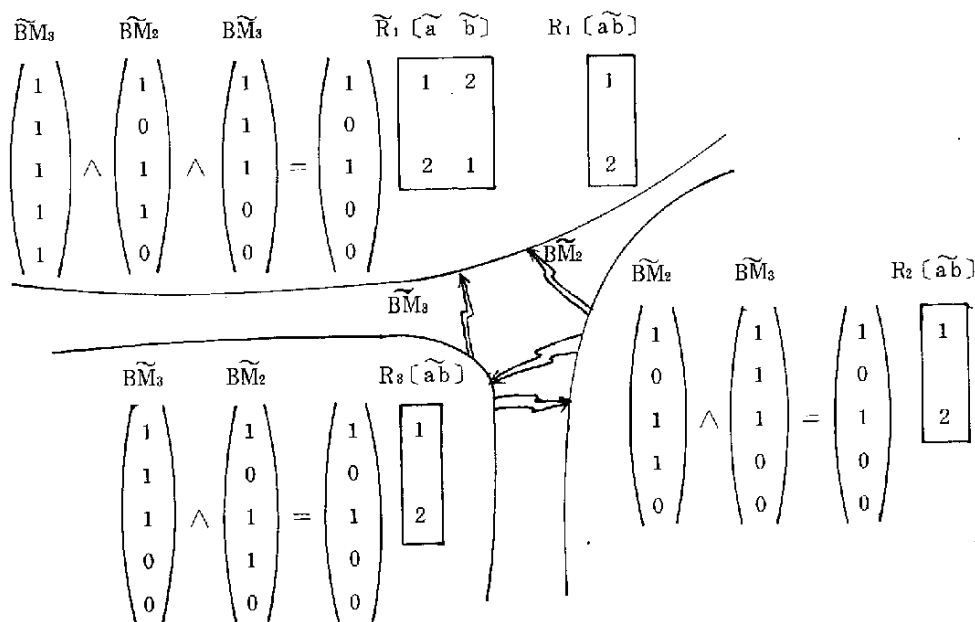


図 2.1 3 BMの放送とコード化

G. 例

例として、図 2.1 4 を考えよう。3つのデータモジュール DM_1 , DM_2 , DM_3 が各々関係 $R_1(a, b)$, $R_2(a, b, c)$, $R_3(b, c, d)$ が存在している。

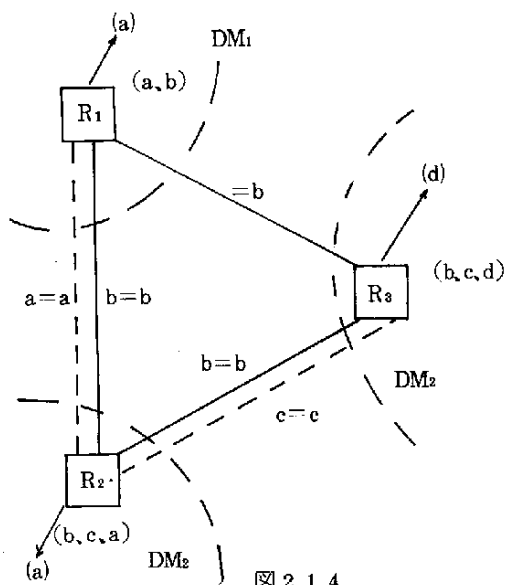


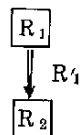
図 2.1 4

(1) 各 $DM_1, \dots, DM_3$ は $R_1 [a], R_1 [b], R_2 [b], R_2 [c], R_2 [d], R_3 [b], R_3 [c]$ の中で最少のものを選ぶ。 $R_1 [a]$ とする。

(2) DM_1 は $R'_1 (= R_1 [a])$ を放送する。

DM_2 では $R_2 \leftarrow R_2 [a=a] R_1$;

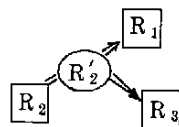
$BM_2 \leftarrow R_2 [a] // R'_1$; BM_2 を R_1 へ送る。



(3) R_1, R_2 の属性 a をコード化する。 $R_1 (\tilde{a}, b, a), R_2 (\tilde{a}, b, c, a)$

(4) 次に、 $R_1 [b], R_2 [b], R_3 [b], R_2 [c], R_3 [c]$ が最少のものを選ぶ。 $R_2 [b]$ とする。

DM_2 は、 $R'_2 \leftarrow R_2 [b]$; を放送する。

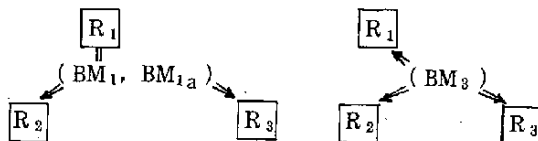


DM_1 では、 $R_1 \leftarrow R_1 [b=b] R'_2$; $BM_{1b} \leftarrow R_{1b} [b] // R'_2$;

BM_{1a} を $R_1 [\tilde{a}]$ より生成

DM_3 では、 $R_3 \leftarrow R_3 [b=b] R'_2$; $BM_{3b} \leftarrow R_{3b} [b] // R'_2$;

DM_1 では、 (BM_1, BM_{1a}) を、 DM_3 では、 (BM_2) を応答として放送する。



(5) DM_1 $BM_{1b} \leftarrow BM_{1b} \wedge BM_{3b}$;

DM_3 $BM_{3b} \leftarrow BM_{3b} \wedge BM_{1b}$;

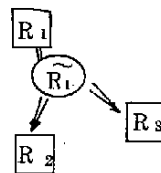
DM_2 $BM_{2b} \leftarrow BM_{1b} \wedge BM_{3b}$;

$BM_{2a} \leftarrow BM_{1a}$;

各 DM で属性 b のコード化を行う。

(6) $R_1 [\tilde{a}, \tilde{b}], R_2 [\tilde{a}, \tilde{b}], R_2 [c], R_3 [\tilde{b}], R_3 [c]$ の中で、最少のものを選ぶ。 $R_1 [\tilde{a}, \tilde{b}]$ が最少とすると、

DM_1 は、 $\tilde{R}_1 \leftarrow R_1 [\tilde{a}, \tilde{b}]$; **broadcast** $\tilde{R}_1$;



(7) DM_2 $R_2 \leftarrow R_2 [\tilde{a}=\tilde{a} \wedge \tilde{b}=\tilde{b}] \tilde{R}_1$;

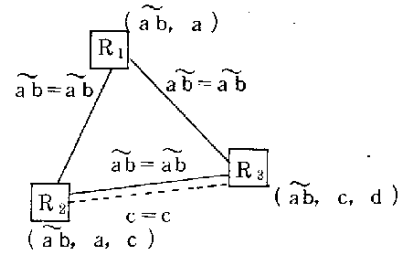
$BM_2 \leftarrow R_2 [\tilde{a}, \tilde{b}] // \tilde{R}_1$; **broadcast** BM_2 ;

DM_3 $R_3 \leftarrow R_3 [\tilde{b}=\tilde{b}] \tilde{R}_1$; $(R_3 (\tilde{a}, \tilde{b}, c, d))$

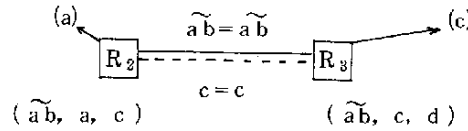
$BM_3 \leftarrow R_2 [\tilde{a}, \tilde{b}] // \tilde{R}_1$; **broadcast** BM_3 ;

- (8) DM₁ BM₁ ← BM₂ ∧ BM₃ ;
 DM₂ BM₂ ← BM₂ ∧ BM₃ ;
 DM₃ BM₃ ← BM₂ ∧ BM₂ ;

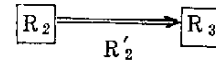
$\tilde{a}$, $\tilde{b}$ を、属性 $\tilde{a}\tilde{b}$ としてコード化する。



この時R₁は、全ての結合属性がコード化され、目標属性aはR₂も有しているので、R₁を消去する。



- (9) R₂[c], R₃[c]のうちで、小さいものを放送する。|R₂[c]| < |R₃[c]|
 とする。DM₂は、R'₂ (= R₂[c])を放送する。



- DM₃で、R₃ ← R₃[c=c]R'₂ ; BM_{3c} ← R₃[c] // R'₂ ;
 BM_{3a~b} ← R₃[a~b]よりbit-mapの生成 ;

broadcast (BM_{3c}, BM_{3a~b}) ;

- (10) DM₂で、BM_{2c} ← BM_{3c} ; BM_{2a~b} ← BM_{3a~b} ;
 (11) DM₂, DM₃で、cのコード化 (R₂ (a~b, c, a), R₃ (a~b, c, b)) ;
 (12) R₂(a~b, c)とR₃(a~b, c)の小さいものを放送する。

DM₃がR'₃ (= R₃[a~b, c])をbroadcastするとする。



- (13) DM₂で、R₂ ← R₂(a~b=a~b ∧ c=c)R'₃ ;
 BM₂ ← R₂(a~b, c) // R'₃ ; broadcast BM₂ ;
 (14) DM₃で、BM₃ ← BM₂ ;
 (15) DM₃, DM₂でa~b, cをa~bcとコード化する。

結果DMに、R₂(a~bc, a) R₃(a~bc, c)を送る。結果DMは最終結果Rを生成する。
 R ← (R₂[a~bc = a~bc] R₃) [a, c] ;

2.4 集積関数 (aggregate function) の処理

集積関数には、次の2種がある。

- (1) 単純集積関数
 - (2) group-by 集積関数
- (1)の単純集積関数は、ある条件によって導出された集合に対する集積演算 (aver, sum,

count, max, min, any)を行い、結果としてスカラー量を得るものである。従って関数内で独立して処理が行える。これに対して、(2)の group-by 集積関数は、グループ化変数の値ごとに集積演算を行い結果を求める。従って、関数内で閉じて処理を行えない。式(1)、(2)は各々(1)、(2)の集積関数の構文を示している。

$$\underline{\text{ag}} \left(\text{exp} [uv_1, \dots, uv_m] [\underline{\text{where}} \quad \text{qual}] \right) \dots\dots\dots (1)$$

$$\underline{\text{ag}} \left(\text{exp} [uv_1, \dots, uv_m] \underline{\text{by}} \quad gv_1 \dots \underline{\text{by}} \quad gv_l \right. \\ \left. [\underline{\text{where}} \quad \text{qual}] \right) \dots\dots\dots (2)$$

ここで $uv_i (i=1, \dots, m) (m \geq 0)$ をユニーク変数 (u変数)、 $gv_j (j=1, \dots, l) (l \geq 1)$ をグループ化変数 (G変数) と呼ぶ。集積関数の意味については第Ⅳ部を参照されたい。

分散問合せ処理 (DQP) では、集積関数を更に次の2点で分類して考える必要がある。

(i) DM内集積関数

(ii) DM間集積関数

(i)は、集積関数内の全ての変数が、同一のLCS内の関係を参照している集積関数である。

(ii)は、複数のLCSにまたがっているものである。

(i)は、LCS内でローカルに、LDPによって処理出来る。

まず、 α を任意の式とした時、以下を定義する。

$$\text{dm}(\alpha) \triangleq \alpha \text{ 内の変数の関係の存在する DM の集合}$$

$$\text{var}(\alpha) \triangleq \alpha \text{ 内の変数の集合}$$

$$\text{vatt}(\alpha) \triangleq \alpha \text{ 内の属性変数の集合}$$

$$\text{ags}(\alpha) \triangleq \alpha \text{ 内の集積関数の集合}$$

$$\text{ag-free}(\alpha) \triangleq \begin{cases} \text{true} & \text{if } |\text{ags}(\alpha)| = 0 \quad (\alpha \text{ は集積関数をもたない}) \\ \text{false} & \text{otherwise} \quad (\alpha \text{ は集積関数を含む}) \end{cases}$$

各集積関数 g に対して

$$\text{gmode}(g) \triangleq \begin{cases} S & \text{if } g \text{ は単純集積関数} \\ G & \text{otherwise} \quad (g \text{ は group-by 集積関数}) \end{cases}$$

$$\text{gvatt}(g) \triangleq g \text{ が group-by 集積関数の時、グループ化(G)変数の集合}$$

$$\text{gvatt}(\alpha) \triangleq \{ \text{gvatt}(g) \mid g \in \text{ags}(\alpha) \wedge |\text{dm}(g)| \geq 2 \} \\ \triangleq \alpha \text{ 内の DM 間 group-by 集積関数 } g \text{ のグループ化(G)変数の集合}$$

$$\text{fvatt}(\alpha) \triangleq \alpha \text{ 内の DM 間集積関数外の属性変数の集合}$$

$$\text{gag}(\alpha) \triangleq \{ g \mid g \in \text{ags}(\alpha) \wedge |\text{dm}(g)| \geq 2 \wedge \text{gmode}(g) = G \} \\ \triangleq g \text{ 内の DM 間 group-by 集積関数の集合}$$

$$\text{sag}(\alpha) \triangleq \{ g \mid g \in \text{ags}(\alpha) \wedge |\text{dm}(g)| \geq 2 \wedge \text{gmode}(g) = S \} \\ \triangleq g \text{ 内の DM 間単純集積関数の集合}$$

問合せQ

var ($l_1, L_1:d_1$).....($l_m, L_m:d_m$);

Q: retrieve into R($a_1=exp_1, \dots, a_k=exp_k$) where $c_1 \wedge \dots \wedge c_n$;

Qに対して、

$TL_Q \triangleq \{exp_1, \dots, exp_k\}$

$QL_Q \triangleq \{c_1, \dots, c_n\}$

[処 理 概 要]

(1) $TL_Q' \leftarrow QL_Q' \leftarrow \phi$; $TL \leftarrow TL_Q$; $GV \leftarrow SA \leftarrow GA \leftarrow \phi$;

[DM間集積関数を含む $exp \in TL_Q$ の分離]

(2) for $\forall exp \in TL_Q$

do

if $|dom(exp)| = 1 \vee$

$(|dom(exp)| \geq 2 \wedge \underline{ag-free}(exp))$

then [exp は DM内の式か、又は exp は集積関数を含まない]

begin $TL_Q' \leftarrow TL_Q' \cup \{exp\}$; $TL \leftarrow TL - \{exp\}$ end;

[DM間集積関数を含む $c \in QL_Q$ の分離]

(3) for $\forall c \in QL_Q$ do

if $|dom(c)| = 1 \vee$

$(|dom(c)| \geq 2 \wedge \underline{ag-free}(c))$

then begin

$QL_Q' \leftarrow QL_Q' \cup \{c\}$; $QL_Q \leftarrow QL_Q - \{c\}$ end;

(4) for $\forall exp \in TL$ (ここで $\underline{ag-free}(exp) = false$ exp は DM間集積関数を含んでいる。)

do begin

$TL_Q' \leftarrow TL_Q' \cup \{\underline{fvatt}(exp)\}$;

$GV \leftarrow GV \cup \{\underline{fvatt}(exp)\}$;

$GA \leftarrow GA \cup \{g \mid g \leftarrow \underline{ags}(exp) \wedge \underline{gmode}(g) = G\}$;

$SA \leftarrow SA \cup \{g \mid g \leftarrow \underline{ags}(exp) \wedge \underline{gmode}(g) = S\}$

end;

(5) for $\forall c = (exp_1 \theta exp_2) \in QL_Q$ ($\underline{ag-free}(c) = false$, c は DM間集積関数を持つ)

do for $\forall i \in \{1, 2\}$ do

```

if ag-free (expi) then TLQ' ← TLQ' ∪ {expi}
else begin TLQ' ← TLQ' ∪ {fvar (expi)};
          GV ← GV ∪ {gvali(expi)};
          GA ← GA ∪ {g | g ∈ ags (expi) ∧ gmode (g) = G};
          SA ← SA ∪ {g | g ∈ ags (expi) ∧ gmode (g) = S};
          end ;

```

ここまでで、GAとSAは各々Q木を前置順にサーチした時の出現順にDM間単純集積関数とgroup-by 集積関数へのポインタが成るリストである。

(6) 全てのsa ∈ SAに対して、一意な変数(PV)を割り当てる。

置換 $\sigma_{sa} \triangleq \{PV_{sa}/sa \mid sa \in SA\}$ によって、

```

TLQ' ←  $\sigma_{sa}$  TLQ' ;          TLQ ←  $\sigma_{sa}$  TLQ
QLQ ←  $\sigma_{sa}$  QLQ ;
GA ←  $\sigma_{sa}$  GA ;
GA ←  $\sigma_{sa}$  (SA - {sa}) ∪ {sa} ;

```

(7) TL_{Q'} = {Q_{exp₁}, ..., Q_{exp_l}} とする。各Q_{exp_i} に対して、一意な属性Q'.b_i を設ける。

置換 $\sigma_{Q'} \triangleq \{Q'.a_i/Q'_{exp_i} \mid Q'_{exp_i} \in TL_{Q'}\}$ によって

```

TLQ'' ←  $\sigma_{Q'}$  · TLQ ;      QLQ'' ←  $\sigma_{Q'}$  · QLQ ;
Q''  $\triangleq$  (TLQ', QLQ') とする。

```

(8) for $\forall ga \in GA$ do

begin

[gaの結果関係スキーム $R_{ga}(va, gv_1, \dots, gv_{n_{ga}})$ を定める。
 [ここで、gaのgroup-by 変数を $gga.v_1, \dots, gga.v_{n_{ga}}$ であるとする。]

[置換 $\sigma_{ga} \triangleq \{R_{ga}.va/ga\}$]

QL_{Q''} ← σ_{ga} · QL_{Q''} ∪ {gga.v_i = R_{ga}.gr_i | i = 1, ..., n_{ga}}

end ;

(9) 各sa ∈ SA (sa (exp, uv₁, ..., uv_l where qual)) に対して

検索 retrieve into R_{sa} (t₀ = exp, t₁ = uv₁, ..., t_l = uv_l) where qual ; を生成し実行させる。結果R_{sa} (t₀, t₁, ..., t_l) に対して集積演算sa を行い解(v_{sa}) を求める。

置換 $\sigma_{vsa} = \{v_{sa}/PV_{sa}\}$

```

GA ←  $\sigma_{vsa}$  · GA ; (TLQ', QLQ') ←  $\sigma_{vsa}$  · (TLQ', QLQ') ;
                  (TLQ'', QLQ'') ←  $\sigma_{vsa}$  · (TLQ', QLQ') ;

```

[単純集積関数(sa)の結果値(v_{sa})によって、問合せ内の sa を置換する]

(0) 各 $ga \leftarrow GA(ga(\text{exp}, uv_1, \dots, uv_l \text{ by } gv_1 \dots \text{by } gv_n \text{ where qual}))$

に対して、

検索 retrieve into $TR_{ga}(t_0 = \text{exp}_1, t_1 = uv_1, \dots, t_l = uv_l, g_1 = gv_1, \dots, gn = gv_n) \text{ where qual}$; を生成し、実行させる。

この結果 $TR_{ga}(t_0, t_1, \dots, t_l, g_1, \dots, g_n)$ に、group-by 集積演算 ga を適用して、結果関係 $R_{ga}(va, gv_1, \dots, gv_n)$ を得る。

(1) 問合せ $Q' = (TL_{Q'}, QL_{Q'})$ を実行し、解 $R_{Q'}(a_1, \dots, a_l)$ を得る。

(2) $R_{Q'}$ と $\{R_{ga} \mid ga \in GA\}$ に対する問合せ $Q'' = (TL_{Q'}, QL_{Q''})$ を実行し、最終的な解 R を得る。

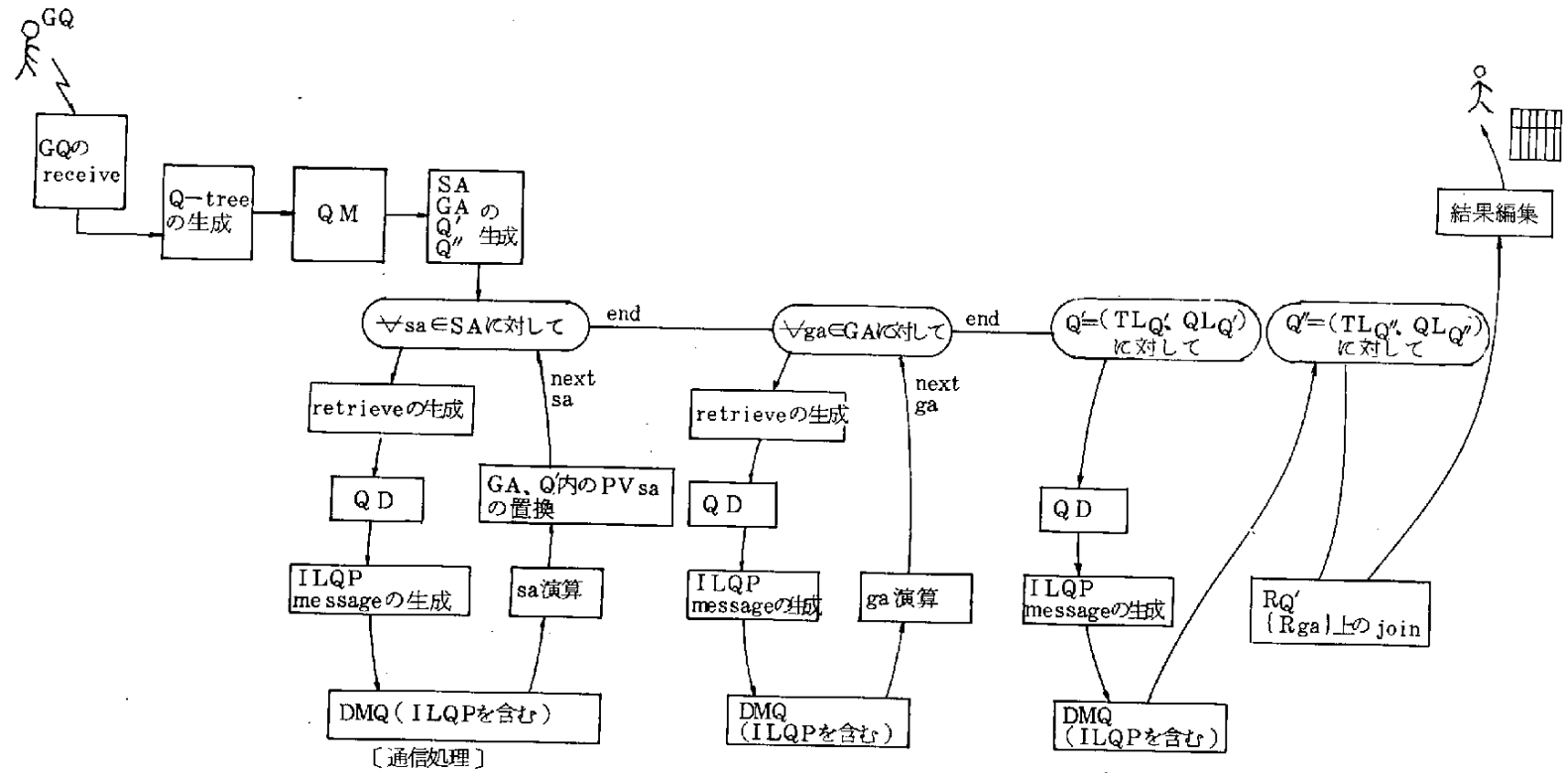


図 2.16 集積関数の処理概要

3. 分散更新処理 (D U P)

分散更新処理 (Distributed update processing (DUP))とは、複数のデータモジュール (DM) 内のデータに対する更新を行うトランザクションを、DB間の通信処理と、DM内のローカル更新によって処理することである。分散更新処理 (DUP) は、複数のトランザクションのもとで、以下の目標を達成する必要がある。

- (i) DM内のデータベースのインテグリティと、DM間のインテグリティを保つ。
- (ii) なるべく多くのトランザクションが、同時実行出来る様にする。

3.1 コミットメント制御

トランザクションが (異なったデータモジュール (DM) に存在するかもしれない) 複数のデータオブジェクトを更新するときに、これらのオブジェクトへの更新が原子的 (atomic) になされることを保障することが、コミットメント制御の目的である。即ち、トランザクション T が、オブジェクト $x_1, \dots, x_n$ ($n \geq 1$) に対する更新を行うときに、 T によって $x_1, \dots, x_n$ への更新が全て行われたか、又は全くなされなかったかにせねばならない [図 3.1]。ここで、 x_i の存在するデータモジュールを DM_i とする。

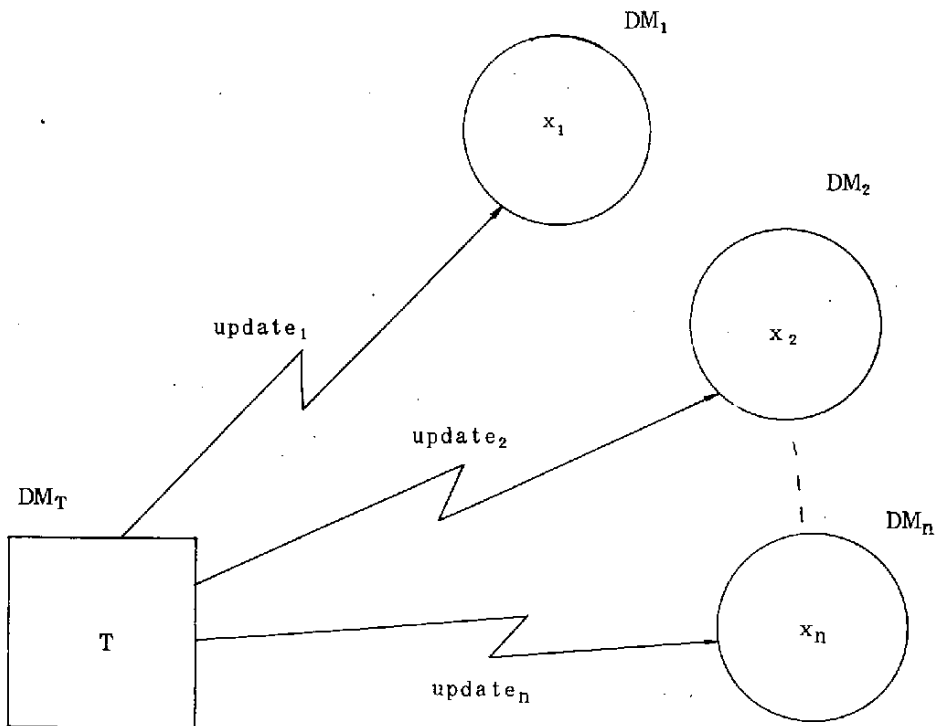


図 3.1 トランザクションとオブジェクト

3.1.1 2相コミットメント制御

コミットメント制御の為の方法としては、2相コミットメント方法〔GRAY〕がある〔図3.2〕。トランザクションTの存在するデータモジュールを DM_T とする。2相コミットメント方法は、次の様をプロトコルに基づいてなされる。

〔2相コミットメント制御方式〕

- (1) オブジェクト $x_1, \dots, x_n$ に対する更新データを、通信処理（アルゴリズムDMQ〔2.3.4〕）によって、 DM_T 内の安全記憶領域（ SS_T ）に生成する。これをRとする（Rを、更新データ集合とする）。
- (2) 各 DM_i に、オブジェクト x_i の更新データ v_i と共に、 $\text{precommit}(x_i, v_i)$ を送る。
- (3) 各 DM_i は、 $\text{precommit}(x_i, v_i)$ を DM_T から受信したのならば、 DM_i の安全な格納領域 SS_i （e.g. ログ）に、 v_i を格納する。 v_i を SS_i に格納出来たならば、 DM_i は DM_T に precommitted を送る。もし失敗すれば、 abort を送る。
- (4) DM_T は、全ての DM_i から precommitted を受信したならば、 DM_T は、各 DM_i に $\text{commit}(x_i)$ メッセージを送る。どれかの DM_j から abort に来るか、又は、応答のないときには、全 DM_j に abort メッセージを送り、トランザクションTをアボート（ abort ）する。
- (5) DM_T から $\text{commit}(x_i)$ を受けた DM_i は、安全格納領域 SS_i の値 v_i を用いて、 DM_i のデータベース DB_i 内のオブジェクト x_i を物理的に更新する。物理的に更新出来たならば、応答 ACK_i を DM_T に返す。各 DM_i は、 precommitted_i を DM_T に送った後、 DM_T から応答がないか、 abort を受ければ、 SS_i 内の値 v_i を消去する。
- (6) DM_T は、全ての DM_i から ACK_i を受けたならば、トランザクションTを正常終了させる。

3.1.2 放送2相コミットメント制御

3.1.1で述べた2相コミットメント制御では、 DM_T と DM 間の1対1通信を基礎にしている。必要なメッセージ数は、(2)の precommit のメッセージ数 n 、(3)の precommit 数が n 、(4)の commit 数が n で、(5)の ACK 数が n で、都合 $4n$ 回の通信が必要になる。これに対して、通信網として、放送網を用いると、通信回数を大幅に縮でる。このアルゴリズムB2Pを以下に示す。

〔アルゴリズム B2P〕

- (1) トランザクションTの更新対象オブジェクト $x_1, \dots, x_n$ ($n \geq 1$) の更新データ ($v_1, \dots, v_n$) をDM間通信処理によって、Tのいるデータモジュール DM_T に生成する。更新データを、 DM_T の安全記憶領域 SS_T （secure storage）に格納する。

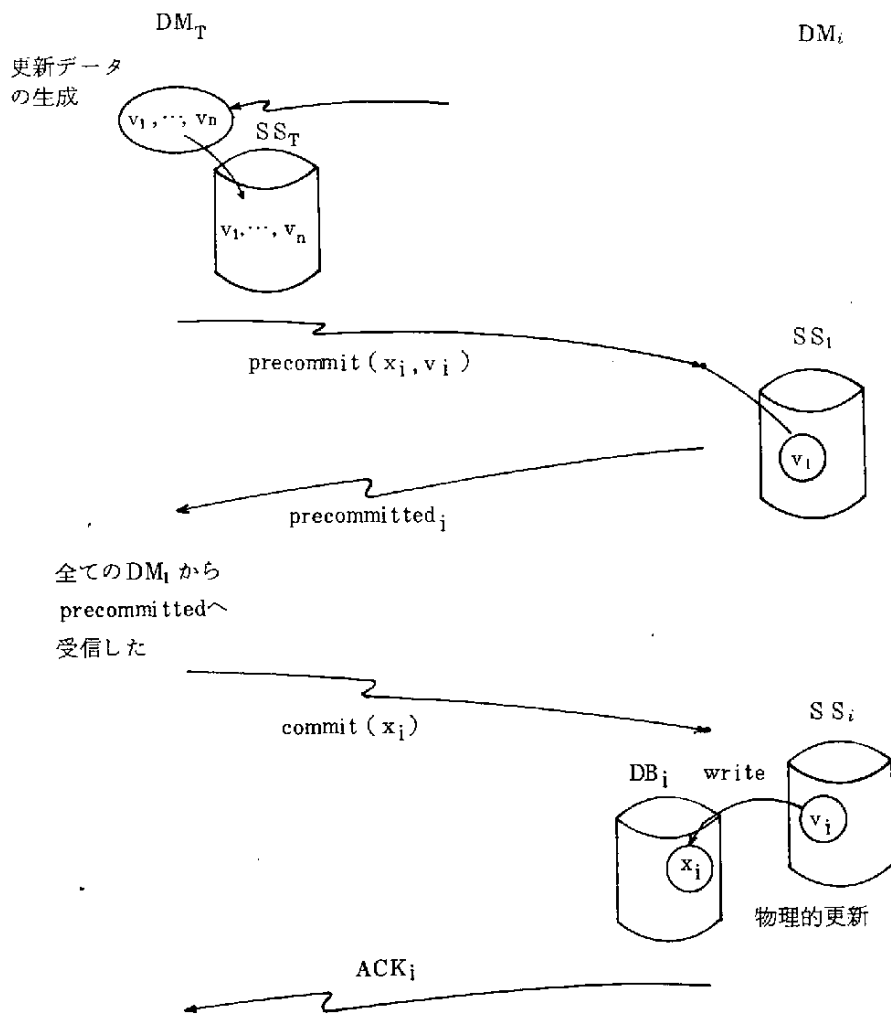


図 3.2 2 相コミットメント

- (2) DM_T は、全 DM_i ($i=1, \dots, n$) に、 SS_T 内の更新データと共に、**precommit** ($v_1, \dots, v_n$) を放送する。
- (3) 各 DM_i は、 DM_T からの **precommit** ($v_1, \dots, v_n$) を受信したならば、オブジェクト x_i の更新データ v_i を、安全格納領域 SS_i に格納する。 v_i を SS_i に格納し終ったならば、**precommitted** (x_i) を放送し、他の全ての DM_j から **precommitted** (x_j) の到着を待つ。
- (4) 各 DM_i は、全ての **precommitted** (x_j) を受信したならば、 SS_i の v_i によって、データベース DB_i 内のオブジェクト x_i を物理的に更新する。物理的に更新されたならば、応答 **ACK_i** (acknowledgement) を放送する。
- (5) DM_T は、全 **ACK_i** を受信したならば、トランザクション T を正常終了させる。

図 3.3 に、アルゴリズム B 2 P を図示する。B 2 P での通信回数を数える。(2)の precommit 数は 1 つ、(3)の precommitted 数は n 、(4)の ACK 数は n であるので、都合の $2n + 1$ 回の通信で、コミットメントの制御を行える。3.1.1 の方式が $4n$ 回の通信を必要とすることに対して、放送網と各 DM の分散制御を用いると $2n + 1$ 回の通信で済み、全通信時間及び応答時間を短縮出来る。

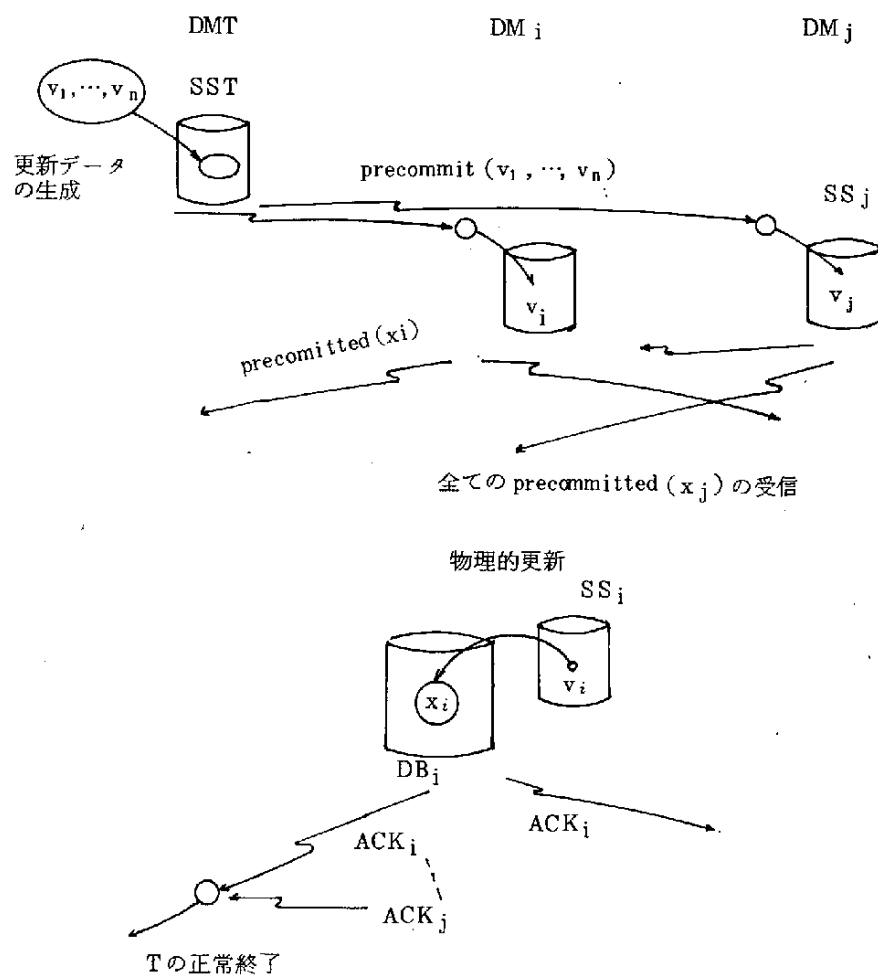


図 3.3 アルゴリズム B 2 P

4. 全体データベースプロセッサ (GDP)

全体データベースプロセッサ (GDP) [TAKIM81, 82c] は、全体概念 (GC) 層と、ローカル概念 (LC) 層間のインタフェースとなり、以下の機能を有している。

- (i) 統合型DDBS設計において、全体管理者 (GA) の入力したGCS定義から、GCSの内部記述と、GCSとLCSとの対応情報を分散情報 (DI) として生成する。このDIを、DDBSの全データモジュール (DM) に送り、その完全コピーを保持させる。
 - (ii) 分散性と異種性に独立なGCS問合せを入力し、以下を行う。
 - (a) GCS問合せを、LCSに基づいた問合せ (全体LCS問合せ) に変換する。
 - (b) 全体LCS問合せを、各DMで独立して処理出来る部分 (LCS問合せ) に分割 (問合せ分割) し、各DMに送り実行させる。
 - (c) 次に、GDP間の通信処理によってDM間の関係の結合処理を行い、解を利用者に出力する。
- (a)を表現変換、(b)を初期ローカル問合せ処理 (ILQP)、(c)をDM間問合せ処理 (DMQP) とする。即ち、分散問合せ処理 (DQP) を行う。

GDPは、DIP (DI processor) とDQP (distributed query processor) との2つのモジュールから成る [図4.1]。DIPは、全体管理者 (GA) によるGCS定義を入力として、分散情報 (DI) を生成し、各DMにDIの完全コピーを配布するプロセッサである。DQPは、分散問合せ処理 (DQP) を行う。DQPは、ETP、QDP、DMQPの3つのモジュールから成る。ETPは、GCS問合せを、LCSを参照する全体LCS問合せ (GLQ) に、DIを用いて行う。即ち、表現変換を行う。QDPは、GLQを、各データモジュール (DM) で独立して処理出来る部分 (LCS問合せ) $LQ_1, \dots, LQ_n$ に分割する (問合せ分割)。分割された LQ_i に送り、これを LDP_i で実行させる。 LQ_i は、共通操作言語RQLによる関係データ構造上の問合せなので LDP_i によってCOBOL DMLに変換され実行され、解が関係としてRWS $_i$ (関係作業域) に導出される。DMQPは、他のDMQPと協同して、自分のRWS内の関係の転送と、他のDMQPから転送された関係との結合結果をRWSに格納し、いわゆる通信処理を行って、解を利用者に出力する。DQP内のETP、QDP、DMQPは、DIを基にして動作する。

利用者がGCS問合せ (GQ) を入力したGDPを、特に結果GDP (RGDP)、又そのDMを結果DM (RDM) とする。

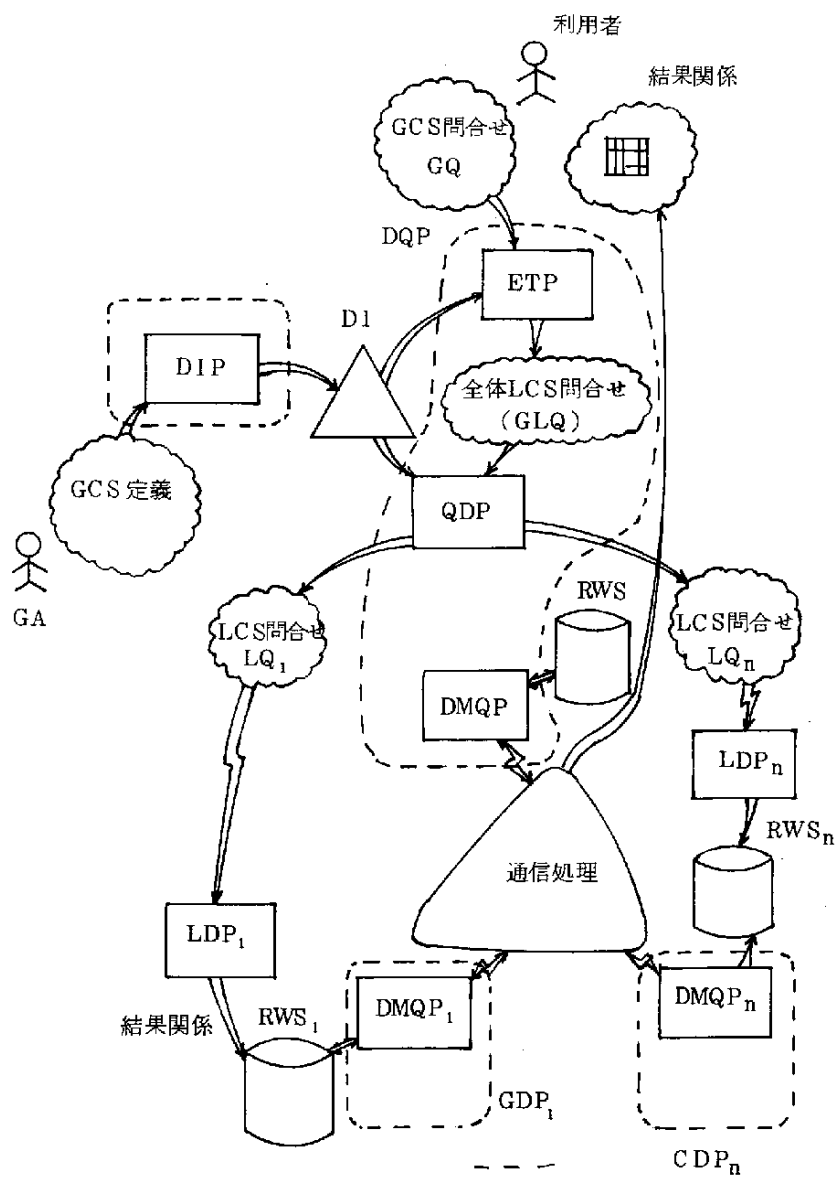


図 4.1 GDP の概要

5. ま と め と 今 後 の 課 題

本部では、異種分散型データベースシステムJ DDBSにおける分散性問題の解決方法を示した。

全体概念(GC)スキーマは、各データモジュール(DM)上のローカル概念(LC)データベース内のLC関係の視野として定義される。

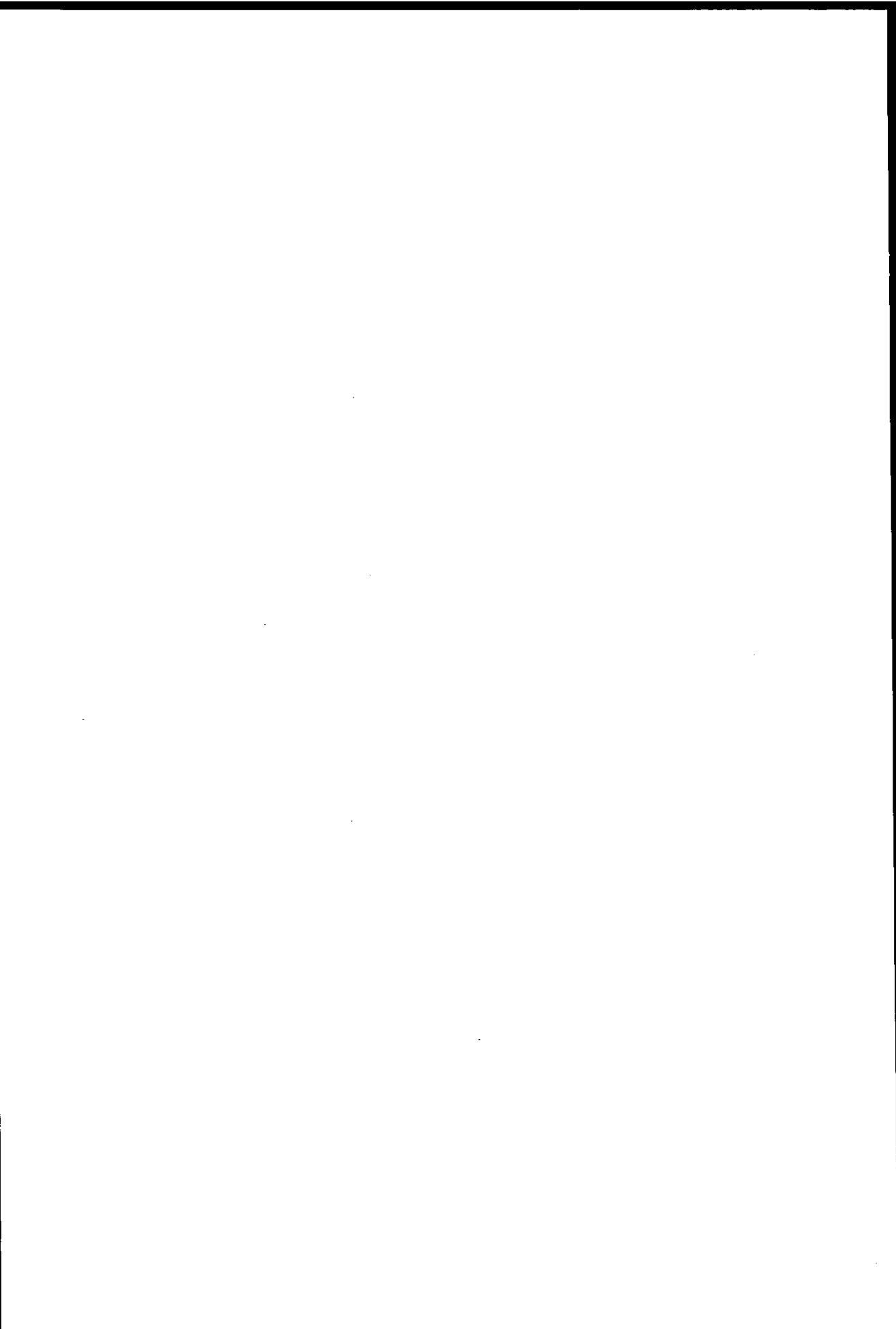
このGCスキーマ上の関係問合せ(RQL問合せ)は、GC問合せである。本部では、GC問合せの分散処理方式を示した。GC問合せは、GC関係スキーマの定義式によって問合せ修正方法によって、LC関係を参照する問合せ(全体LC問合せ(GLQ))に変換する(表現変換)。次にGLQを、各DMで実行できる部分をRQLによるLC問合せとして分割し、各DMで実行させる(初期ローカル問合せ処理(ILQP))。ILQPの目標は、DM間での問合せ処理の為の通信量の減少とともに、異種のDBSのデータ構造から、共通の関係データ構造を導出することである。ILQP後、DM間での結合を処理する為、DM間での関係の転送と、DM内でのローカル処理が行われる(DM間問合せ処理(DMQP))。DMQPでは、関係の放送と、データ圧縮技法を用いた、完全分散制御と動的スケジュール決定に基づいた一般問合せの分散処理アルゴリズムDMQを示した。

更に、放送通信と分散制御に基づき、従来の2相コミットメント方式の通信回数を大幅に減少させる放送2相コミットメント方法(B2P)を示した。

本部では、放送網を全通信時間と応答時間とから定義し、分散性問題を解決する為のGDPアーキテクチャを示した。本部はGDP実現の論理的基礎となる。

現在、GDPは、放送通信網をM-170F(OS IV/F4)のAIM DB/DCによってシミュレートしているが、現在、実際の放送網(ローカル網(LAN))の導入を検討している。あわせて、パーソナル計算機等の小型システム内でのGDP実現を目指している。又、更新については、GC関係を抽象データ型として、許される更新演算を規定してしまう方法を考えている。

第Ⅳ部 J D D B S の実現



第IV部 JDDBSの実現

本部では、異種分散型データベースシステムJDDBS (Jipdec DDBS) の実現方法、及び実現結果について述べる。第1章では、JDDBSの全体構成を述べる。第2章では、CODASYLデータベースシステム上の共通関係インタフェースとしてのローカルデータベースプロセッサ (LDP) について述べる。第3章では、各データモジュール間の通信処理を行う全体データベースプロセッサ (GDP) について述べる。最後に、JDDBSの研究開発経過を簡単に述べる。

1. JDDBSの全体構成

JDDBSは、現在、図1.1に示す様に、当協会のM-170F内の4つのCODASYL DBSから構成されている。各CODASYL DBSは、AIM DBMSによって管理された次の4つである。

- (1) DCDBS
- (2) PRDBS₁
- (3) PRDBS₂
- (4) PRDBS₃

DCDBSは、プロジェクトで購入した書籍資料についての情報を有している。PRDBSは、プロジェクトとメンバ、メンバの書いたレポートについての情報を有している。

JDDBSは、通信網 (CN) で結合された4つのデータモジュール (DM) から成る。各DMは、(1)~(4)のCODASYL DBSの1つと、ローカルデータベースプロセッサ (LDP)、全体データベースプロセッサ (GDP)、システム格納領域 (SS) から成る。SSは、分散情報 (DI)、異種性情報 (HI)、及び関係作業領域 (RWS) から成る。SSは、1つのAIM DBSとして実現されている。GDPとLDPは、独立したプロセスとして存在していて、各々SS内のDIとHIを用いて動作する。通信網CN及び、各プロセス間の通信は、AIMのDB/DCを用いて、放送通信をシミュレートしたものを設定している。

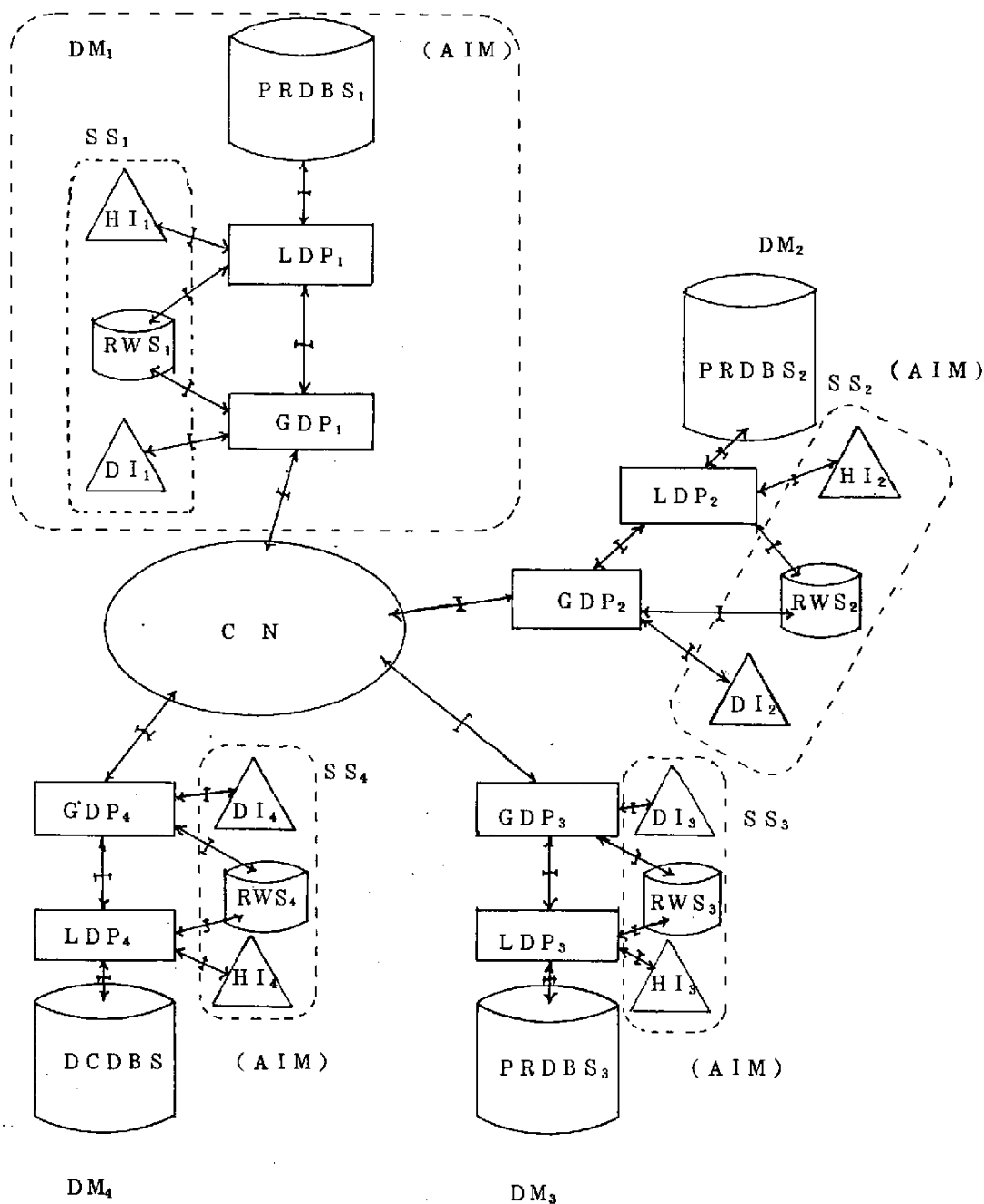


図 1.1 JDDBS の構成

1.1 データベースシステム (DBS)

JDDBSを構成するCODASYL (AIM) DBSを説明する。各PRDBS_iは、同一のスキーマを有し、同一内容のデータベース上に、異なったサブスキーマを持つ様につくられている。よって、DCDBSと、PRDBSについて述べる。又、各DBSは、互いに異ったディスクドライブ上に設けてある。

1.1.1 AIM DBSの特徴

AIMは、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。しかし、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の標準であるCODASYL仕様 (CODAS 73) と異なっている。ここでは、CODASYL仕様との相違点を明らかにし、AIMの特徴について述べる。

A. データ構造

AIMでは、CODASYL仕様にあるメンバシップクラスのサポートがない。この為、セット関係は利用者の責任で管理されなければならない。

又、データベースへのアクセスを効率的に行なわせる為に、INDEXやSUB RANGE (RANGEの分割) 等の構造を利用者に提供する。

実現値の格納方法としてCALC項目値として、AIMでは重複値を許さない (DNA指定) 場合だけがサポートされる。

B. DML言語

a) サブスキーマ宣言

AIMでは、環境部 (ENVIRONMENT DIVISION) でサブスキーマを宣言する。CODASYL仕様では、DATA DIVISION内のSUB-SCHEMA SECTIONで宣言する。

AIM

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SUB-SCHEMA-NAME "PROJSUB4".

b) READY命令

AIMには、READYと命令でUSAGE-MODEの指定が無い。データベースの共有制御は、ジョブ制御文でシステムに登録されたPED (応用プログラムの動作環境の設定) によって制御される。

c) 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は特別な指定が必要ない。文字タイプの8桁のデータ項目であれば良い。

AIM

03 DBKEY PIC X(08)

d) 現在子

AIMでは、応用プログラム、セット型、レコード型、インデクスの最新状態（現在子）をもつ。これらの最新状態は、AIMと応用プログラム間の連絡領域内に確保される。

<u>AIM</u>	データ項目名
応用プログラムの最新状態	PGCS
セット型の最新状態	STCS
レコード型の最新状態	RDCS
インデクスの最新状態	IXCS

プログラム内では、これらのデータ項目の値（現在子）を save / restore することができる。

e) ACCEPT命令

AIMには、ACCEPT命令がなく、連絡領域内のデータ項目PGCSの値を取り出すことがACCEPT命令に対応する。

d) FIND命令

i) データベースキーによるFIND命令

AIMには、データベースキーによるFIND命令がない。データベースキーを連絡領域のデータ項目PGCSにセットし、FIND CURRENT命令を実行することになる。

AIM

```
MOVE DBKEY TO PGCS OF FCOM.
FIND CURRENT.
```

ii) セット内のFIND命令

AIMでは、セット型がチェーン形態の場合には、次のFIND命令がサポートされない。

AIM

```
FIND { LAST
      FIRST } レコード名 WITHIN セット名.
      整数
```

iii) CALCキーによるFIND DUPLICATE命令

CALCキーに対して重複が許されないの、サポートされない。

f) ERASE命令

AIMでは、ERASE PERMANENTがサポートされない。

g) MODIFY命令

AIMでは、MODIFY命令でセット関係を変更できない。レコード型の項目の内容だけしか変更できない。

h) Precompile 方式

AIMでは、COBOL DMLはPrecompile方式である。

1.1.2 DCDBS(ドキュメントDBS)

DCDBSは、プロジェクトの購入した図書についての情報を有している。DCDBSのスキーマを図1.2に示す。

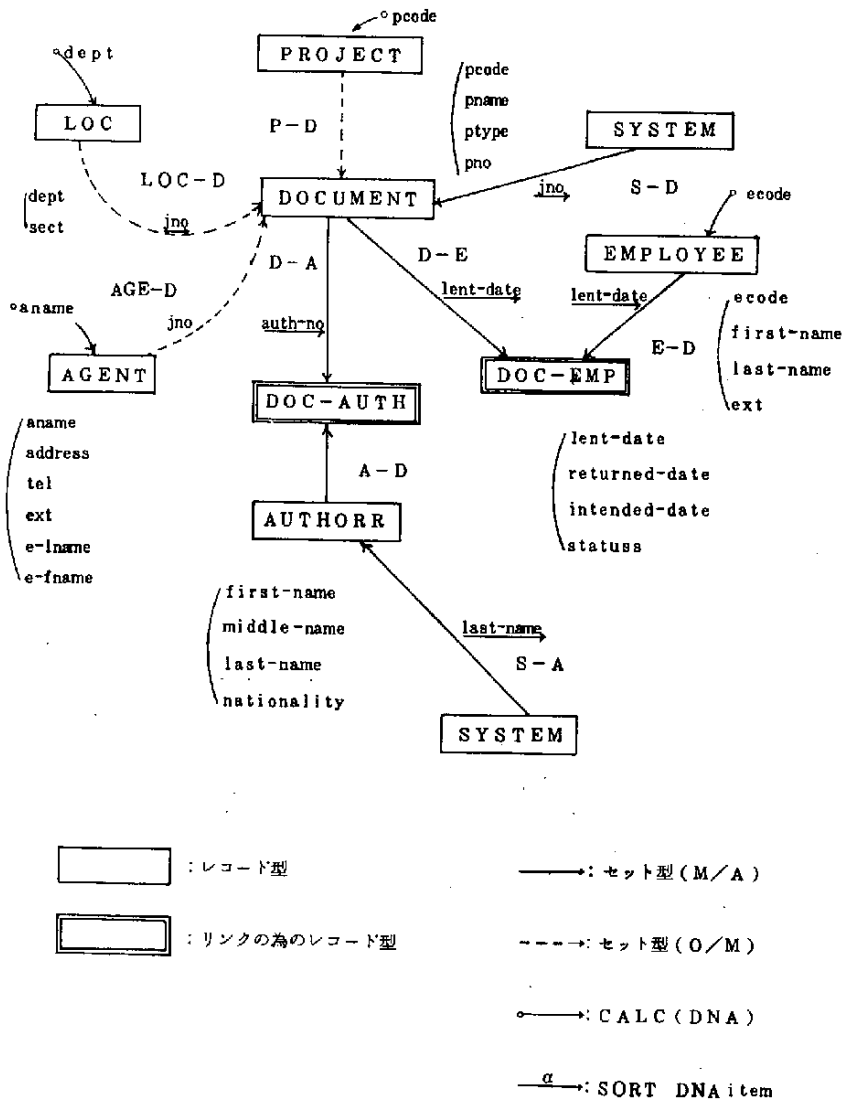


図1.2 DCDBSのLIS(CODASYL スキーマ)

A. レコード型の項目内容

DCDBSのレコード型について説明する。

(1) DOCUMENT 図書情報を表す。

注) NNA:省略不可項目 (APPEND時)

NA :省略可能項目

項 目 名	タイプ	桁 数	NNA/NA	例	説 明
jno	I	7	NNA	4335	受入登録番号 洋書 0~99999 和書 100001~199999 雑誌 200001~299999
isbn	C	20	NA	'0914894137'	US ISBN 0-914894-13-7
c-code	I	4	NA	433	分類コード (ACM分類コード)
c-descript	C	10	NA	'E27-56'	分類
typ	C	1	NA	'M'	M:雑誌 B:書籍
title	C	400	NA	'DATA MODEL'	書名
no-of-author	I	2	NA	1	著者数
p-year	I	2	NA	81	出版(年)
p-month	I	2	NA	10	出版(月)
p-company	C	100	NA	'JIPDEC'	出版社名
cost	I	8	NA	16000	価格(円)
purch-year	I	2	NA	81	納入(年)
purch-month	I	2	NA	06	納入(月)

項 目 名	タイプ	桁 数	NNA/NA	例	説 明
purch-day	I	2	NA	15	納入(日)
fiscal-year	I	2	NA	82	会計年度
order-year	I	2	NA	82	発注(年)
order-month	I	2	NA	06	発注(月)
order-day	I	2	NA	24	発注(日)
orde	I	5	NA	00201	注文番号
code-in-proj	C	10	NA	'56-102'	project内番号
vol	C	6	NA	'SE-3'	巻
numb	I	4	NA	1	番号
tpage	I	5	NA	100	全ページ数
language	C	1	NA	'E'	J:日本語 E:英 語 F:仏 語 G:独 語

(2) AGENNT 図書の納入業者を表す。

項 目 名	タイプ	桁 数	NNA/NA	例	説 明
aname	C	100	NNA	'JIPDEC'	代理店名
address	C	100	NA	'MINATO-KU'	住 所
z-code	C	6	NA	'160'	郵便番号 ×××-××
tel	C	12	NA	'03-434 -8211'	電話番号 03-×××-×××× ××××-××-××××
ext	C	4	NA	'3851'	内線番号
e-lname	C	30	NA	'YAMADA'	担当者(名字)
e-fname	C	30	NA	'JIRO'	" (名前)

(3) PROJECT 図書を購入したプロジェクト情報を示す。

項 目 名	タイプ	桁 数	NNA/NA	例	説 明
pcode	I	4	NNA	0110	業務コード
pname	C	100	NA	'Man Machine User interface'	業務名
ptype	C	2	NA	'H'	プロジェクトタイプ { 補助金 H 受託 J 自主
syear	I	2	NA	82	開始年度
eyear	I	2	NA	85	最終年度
pno	C	8	NA	'MMUI82'	PRDBSのpno

(4) AUTHOR 著者の情報を示す。

項 目 名	タイプ	桁 数	NNA/NA	例	説 明
first-name	C	30	NNA	'E.'	名前(省略形)
middle-name	C	10	NA	'F.'	(省略形)
last-name	C	30	NA	'CODD'	氏(full)
nationality	C	2	NA	'US'	国籍 US:アメリカ BR:英国 JA:日本

(5) DOC-EMP DOCUMENTとEMPLOYEE間のリンクの為のレコード型である。

図書の貸出し、返却情報を表す。

項 目 名	タイプ	桁 長	NNA/NA	例	説 明
lent-date	I	6	NNA	820615	貸出日(年月日)
return-year	I	2	NA	82	返却(年)
return-month	I	2	NA	6	返却(月)
return-day	I	2	NA	26	返却(日)
intend-year	I	2	NA	82	返却予定(年)
intend-month	I	2	NA	6	返却予定(月)
intend-day	I	2	NA	20	返却予定(日)
status	C	2	NA	'0'	返却状態 0 貸出し中 1 返却(正常) 2 " (異常)

(6) LOC 図書の所在情報。

項 目 名	タイプ	桁 長	NNA/NA	例	説 明
dept	C	2	NNA	'D'	部 名 D:開発部 R:技術調査部 A:総務部 M:マイコンセンター I:IIT
sect	C	3	NA	'D1'	課 名 D1:開発第1課 D2: " 2課 D3: " 3課 :
mname	C	30	NA	'SUGA'	本の管理者 (名 字)
ext	I	4	NA	210	mnameの内線番号

(7) EMPLOYEE 従業員情報を表す。

項 目 名	タイプ	桁 長	NNA/NA	例	説 明
ecode	I	4	NNA	0272	職員番号
first-name	C	30	NA	'MAKOTO'	名 前
last-name	C	30	NA	'TAKIZAWA'	名 字
dept	C	2	NA	'D'	部 名 D:開発部 R:技術調査部 A:総務部 M:マイコンセンター I:IIT
ext	I	4	NA	213	内線番号
eno	I	5	NA	1	PRDBSのeno

(8) DOC-AUTH DOCUMENTとAUTHORR間のリンクの為のレコード型である。図書と著者の関係を表す。

項 目 名	タイプ	桁 数	NNANA	例	説 明
auth-no	I	2	NNA	1	著者番号

B. セット型の概要

DCDBSのセット型について説明する。

(1) S-D

レコード型DOCUMENTを子レコードとして持つ単項セット型である。メンバシップクラスはMANDATORY AUTOMATIC (M/A)である。項目JNOで昇順に並べられる。項目JNOの値は重複値が許されない。

(2) S-A

レコード型AUTHORRを子レコードとして持つ単項セット型である。メンバーシップクラスはM/Aで、項目LAST-NAMEで昇順に並べられる。項目LAST-NAMEの値は重複値が許されない。

(3) LOC-D

図書(DOCUMENT)と、保管場所(LOC)の間の関係を表わし、項目JNOの昇順に並べたセット型である。メンバシップクラスは、OPTIONAL MANUAL (O/M)である。項目JNOの値は重複値が許されない。

(4) AGE-D

代理店(AGENT)と、その代理店から購入した図書(DOCUMENT)との間の関係を表わし、項目JNOの昇順に並べたセット型である。メンバシップクラスはO/Mである。項目JNOの値は重複値が許されない。

(5) P-D

プロジェクト(PROJECT)と図書(DOCUMENT)との間の関係を表わし、項目JNOの昇順に並べたセット型である。メンバシップクラスは、O/Mである。項目JNOの値は重複値が許されない。

(6) E-D

従業員(EMPLOYEE)と図書(DOCUMENT)とのn:mの関係を表わす為、従業員(EMPLOYEE)とリンクレコード型(DOC-EMP)との関係を表わすセット型である。メンバシップクラスは、M/Aである。項目LENT-DATEの昇順に並べられる。

(7) D-E

図書(DOCUMENT)と従業員(EMPLOYEE)とのn:mの関係を表わす為、図書(DOCUMENT)とリンクレコード型(DOC-EMP)との関係を表わすセット型である。メンバシップクラスは、M/Aである。項目LENT-DATEの昇順に並べられる。項目LENT-DATEの値は重複値が許されない。

(8) D-A

図書(DOCUMENT)と著者(AUTHORR)とのn:mの関係を表わす為、図書(DOCUMENT)とリンクレコード型(DOC-AUTH)との関係を表わすセット型

である。メンバシップクラスは、M/Aである。項目AUTH-NOの昇順に並べられる。
項目はAUTH-NOの値は重複値が許されない。

(9) A-D

著者(AUTHORR)と図書(DOCUMENT)とのn:mの関係を表わす為、著者(AUTHORR)とリンクレコード型(DOC-AUTH)との関係を表わすセット型である。メンバシップクラスは、M/Aである。

C. DCDBSのデータベース容量

DCDBSは、8のレコード型と9つのセット型から成る。各レコード型、セット型の実現値数は、表1.1、1.2の様になっている〔'83 2月の時点〕。

表1.1. レコード実現値数

レコード型名	リンクレコード型 (yes/NO)	実現値数
DOCUMENT	NO	279
AUTHORR	NO	268
AGENT	NO	23
EMPLOYEE	NO	10
LOC	NO	1
PROJECT	NO	2
DOC-AUTH	yes	331
DOC-EMP	yes	75

表1.2 セット実現値数

セット型名	セット実現値数
P-D	2
LOC-D	1
AGE-D	12
D-A	202
A-D	256
D-E	36
E-D	10
S-D	1
S-A	1

1.1.3 PRDBS (プロジェクトDBS)

PRDBSは、プロジェクトと従業員、従業員と所属する組織、従業員とその著者等の情報を有している。PRDBSのスキーマを図1.3に示す。

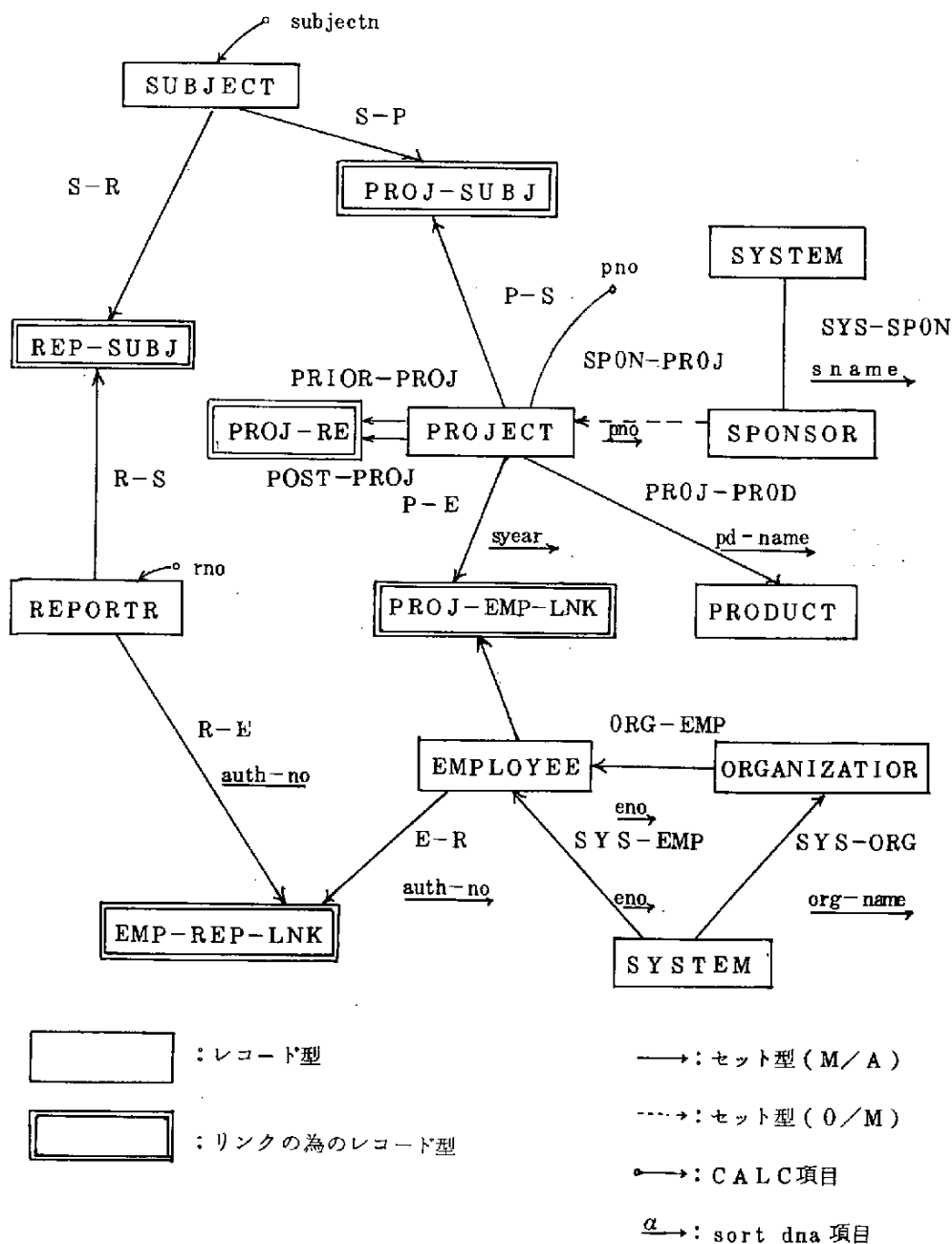


図 1.3 PRDBSのLIS

A. レコード型の項目内容

PRDBSのレコード型について説明する。

(1) EMPLOYEE 従業員情報を表す。

注) NNA : 省略不可項目 (APPEND)
NA : 省略可能項目 時

項 目 名	タイプ	桁 数	NNA/NA	例	説 明
eno	I	5	NNA	1	従業員番号
ename	C	10	NA	'Takizawa'	名 字
efname	C	30	NA	'Makoto'	名 前
esex	C	1	NA	M	性 別 M : 男 F : 女
birth-place	C	10	NA	'TOKYO'	出身地
birth-year	I	2	NA	5 0	生年月日 (年)
birth-month	I	2	NA	1 2	生年月日 (月)
alma-mater	C	30	NA	'TOHOKU UNIVERSITY'	出身校
major	C	30	NA	'PHYSICS'	学 部
degree	C	1	NA	'M'	M : Master D : Doctor B : Bachelor
deg-year	I	2	NA	7 3	卒業年度
hired-year	I	2	NA	7 3	入社年度
retired-year	I	2	NA		退社年度
dept	C	20	NA	DEVELOP MENT'	部 名
eposition	C	20	NA	PROJECT LEADER'	職 種
tel	C	12	NA	'03-434 -8211'	電話番号

(2) PROJECT プロジェクト情報を表す。

項 目 名	タイプ	桁 数	NNA/NA	例	説 名
pno	C	8	NNA	'MMUI80'	プロジェクト番号
pname	C	100	NA	'ManMachine User Interface'	プロジェクト名
proj-type	C	1	NA	'R'	プロジェクトタイプ
status-f	C	3	NA	'ON'	実行状況 ON : 実行中 OFF: 終了
syear	I	2	NA	80	開始年度
eyear	I	2	NA	82	終了年度
budget	I	11	NA		予 算

(3) REPORTR レポート情報を表す。

項 目 名	タイプ	桁 数	NNA/NA	例	説 明
rno	C	8	NNA	'TAKIM83B'	レポート番号
title	C	200	NA		タイトル
source-n	C	100	NA	'JIPDEC TR'	
vol	C	8	NA		巻
numb	I	3	NA		番号
month	I	2	NA	1	発行日
year	I	2	NA	83	発行年
from-page	I	5	NA	1	開始ページ番号
to-page	I	5	NA	30	終了ページ数
total-page	I	5	NA	30	トータルページ数
no-of-author	I	2	NA	2	著者名
language	C	10	NA	'ENGLISH'	言 語
written-year	I	2	NA	82	書かれた年
cost	I	9	NA	0	コスト

項 目 名	タイプ	桁 数	NNA/NA	例	説 明
rtype	C	1	NA		
rstate	C	1	NA	'u'	公用(P)or非公用('u')

(4) SPONSOR スポンサー情報を表す。

項 目 名	タイプ	桁 数	NNA/NA	例	説 明
sname	C	20	NNA	'MITI'	スポンサ名
stype	C	10	NA	'PUBLIC'	スポンサタイプ

(5) SUBJECT 主題情報を表す。

項 目 名	タイプ	桁 数	NNA/NA	例	説 明
subject n	C	100	NNA	'DATA MODEL'	主 題

(6) ORGANIZATION 組織情報を示す。

項 目 名	タイプ	桁 数	NNA/NA	例	説 明
org-name	C	50	NNA	'JIPDEC'	組織名
org-type	C	10	NA	'P'	タイプ (P:公共機関)

(7) PRODUCT

プロダクト情報を表す。

項 目 名	タイプ	桁 数	NNA/NA	例	説 明
pd-no	C	5	NNA	'LDP82'	プロダクト番号
pd-name	C	30	NA	'LDP-V3'	プロダクト名
ptype	C	1	NA	'R'	プロダクトタイプ
cost	I	9	NA	0	コスト
year	I	2	NA	57	年 度
language	C	10	NA	'PL/I'	プログラム言語
psize	I	5	NA		プログラムサイズ
os-name	C	15	NA	'OS IV/F4'	OS名

(8) EMP-REP-LNK

EMPLOYEEとREPORTR間のリンクの為のレコード型である。従業員とレポートの関係を表す。

項 目 名	タイプ	桁 数	NNA/NA	例	説 明
auth-no	I	2	NNA	1	著者番号

(9) PROJ-EMP-LNK

PROJECTとEMPLOYEE間のリンクの為のレコード型である。プロジェクトと従業員の関係を表す。

項 目 名	タイプ	桁 数	NNA/NA	例	説 明
statuss	C	4	NA	'ON'	ステータス
syear	I	2	NA	80	開始年
smonth	I	2	NA	04	開始月
eyear	I	2	NA	83	終了年
emonth	I	2	NA	03	終了月
pposition	C	10	NA	'MEMBER'	プロジェクト内での身分
role	C	10	NA		プロジェクト内での役目
percentage	I	3	NA		稼働割合

B. セット型の概要

PRDBSのセット型について説明する。

(1) SYS-SPON

レコード型SPONSORを子レコードとして持つ単項セット型である。

(2) SYS-ORG

レコード型ORGANIZATIONを子レコードとして持つ単項セット型である。

(3) SYS-EMP

レコード型EMPLOYEEを子レコードとし、データ項目(en)に関して、昇順に並べた単項セットである。データ項目(en)の値は重複値が許されない(DNA)。

(4) ORG-EMP

組織とその組織に属する従業員(EMPLOYEE)との関係を表わしたセット型である。データ項目(en)に関して、昇順に並べられる。データ項目(en)の値は、重複値が許されない(DNA)。

(5) E-R

従業員(EMPLOYEE)と、レポート(REPORTR)の間のn:mの関係を表わす為に、従業員とリンクレコード型(EMP-REP-LNK)との関係を表わすセット型。データ項目(auth-no)に関して、昇順に並べられる。

(6) R-E

レポート(REPORTR)と従業員(EMPLOYEE)の間のn:mの関係を表わす為にレポートとリンクレコード型(EMP-REP-LNK)との関係を表わすセット型。データ項目(auth-no)に関して、昇順に並べられる。データ項目(auth-no)の値は、重複値が許されない。

(7) R-S

レポート(REPORTR)と主題(SUBJECT)との間のn:mの関係を表す為にレポートとリンクレコード型(EMP-REP-LNK)との関係を表わすセット型。

(8) S-R

主題(SUBJECT)とレポート(REPORTR)との間のn:mの関係を表わす為、主題(SUBJECT)とリンクレコード型(PROJ-SUBJ)との関係を表わすセット型である。

(9) S-P

主題(SUBJECT)とプロジェクト(PROJECT)との間のn:mの関係を表わす為、主題(SUBJECT)とリンクレコード型(PROJ-SUBJ)との関係を表わすセット型である。

(10) P-S

プロジェクト(PROJECT)と主題(SUBJECT)との間のn:mの関係を表わす為プロジェクト(PROJECT)とリンクレコード型(PROJ-SUBJ)との関係

表わすセット型である。

(11) P-E

プロジェクト (PROJECT) と従業員 (EMPLOYEE) との間の $n:m$ の関係を表わす為、プロジェクト (PROJECT) とリンクレコード型 (PROJ-EMP-LNK) との関係を表わすセット型である。データ項目 (syear) に関して、昇順に並べられる。

(12) E-P

従業員 (EMPLOYEE) とプロジェクト (PROJECT) との間の $n:m$ の関係を表わす為、従業員 (EMPLOYEE) とリンクレコード型 (PROJ-EMP-LNK) との関係を表わすセット型である。データ項目 (syear) に関して、昇順に並べられる。

(13) PRIOR-PROJ

プロジェクトとそのプロジェクトに関係した前のプロジェクトとの関係を表わす為に、プロジェクト (PROJECT) とリンクレコード型 (PROJ-RE) との関係を表わすセット型である。

(14) POST-PROJ

プロジェクトとそのプロジェクトの後続プロジェクトとの関係を表わす為に、プロジェクト (PROJECT) をリンクレコード型 (PROJ-RE) との関係を表わすセット型である。

(15) SPON-PROJ

スポンサ (SPONSOR) とプロジェクト (PROJECT) との関係を表わすセット型である。データ項目 (pno) に関して、昇順に並べられる。データ項目 (pno) の値は、重複値が許されない。

(16) PROJ-PROD

プロジェクト (PROJECT) とそのプロダクト (PRODUCT) との関係を表わすセット型である。データ項目 (pd-no) に関して、昇順に並べられる。データ項目 (pd-no) の値は、重複値が許されない。

C. PRDBSのデータベース容量

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、セット型の実現値数は、表1.3、1.4の様になっている。['83 2月の時点]

表 1.3 レコード型の実現値数

レコード型名	リンクレコード型 (Yes/No)	実 現 値 数
EMPLOYEE	No	77
ORGANIZATIOR	No	18
PRODUCT	No	11
PROJECT	No	15
SUBJECT	No	181
REPORTR	No	102
SPONSOR	No	4
PROJ-EMP-LNK	Yes	72
EMP-REP-LNK	Yes	199
REP-SUBJ	Yes	855
PROJ-SUBJ	Yes	206
PROJ-RE	Yes	5

表 1.4 セット型の実現値数

セ ッ ト 型 名	実 現 値 数
SYS-SPON	1
SYS-ORG	1
SYS-EMP	1
ORG-EMP	18
E-R	46
R-E	100
R-S	100
S-R	123
S-P	104
P-S	15
P-E	15
E-P	37
PRIOR-PROJ	4
POST-PROJ	4
SPON-PROJ	4
PROJ-PROD	3

1.2 システム格納領域(SS)

システム格納領域(SS)は、分散情報(DI)、異種性情報(HI)、及び関係作業領域(RWS)から成っている。SSは、1つのAIM DBSとして実現されている。DI、HIとRWSのCODASYLスキーマは図1.4、1.5の様である。DI、HIとRWSは独立な(セット型でリンクされていない)DBSとなっている。

A. DI

分散情報(DI)は、次の情報からなる。

(1) GCSの記述

GCSの記述は、レコード型GCS、GREL、GATTによって情報が保持される。

(2) GCSとLCSの対応情報

GCSとLCSの対応情報は、リンクレコード型LVAR、レコード型GDEFによって保持される。

(3) LCSの情報

LCSの情報は、レコード型LCS、LREL、LATTに保持される。

(4) GCSユーザ情報

GCSユーザ情報は、レコード型GU、リンクレコード型GUIに保持される。

B. HI

異種性情報(HI)は、次の情報から成る。

(1) LIS情報

LIS情報は、レコード型SETH、RECH、ITEMHに保持される。

(2) LCS及びLCSとLISの対応情報

LCS及びLCSとLISの対応情報に、レコード型ESRH、RSRH、ATTHに保持される。

(3) DML情報

DML情報は、レコード型DMLI、DMLSTに保持される。

C. RWS

関係作業領域(RWS)には、問合せの結果関係の情報を保持する。結果関係の情報は、レコード型REL、ATTR、TUPLE、TVLNK、VAL、VOVERに保持される。

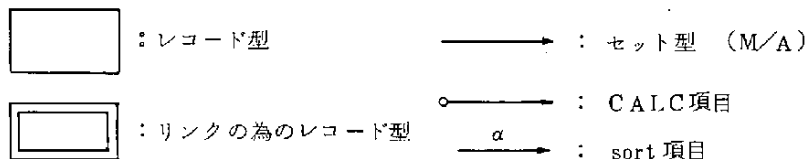
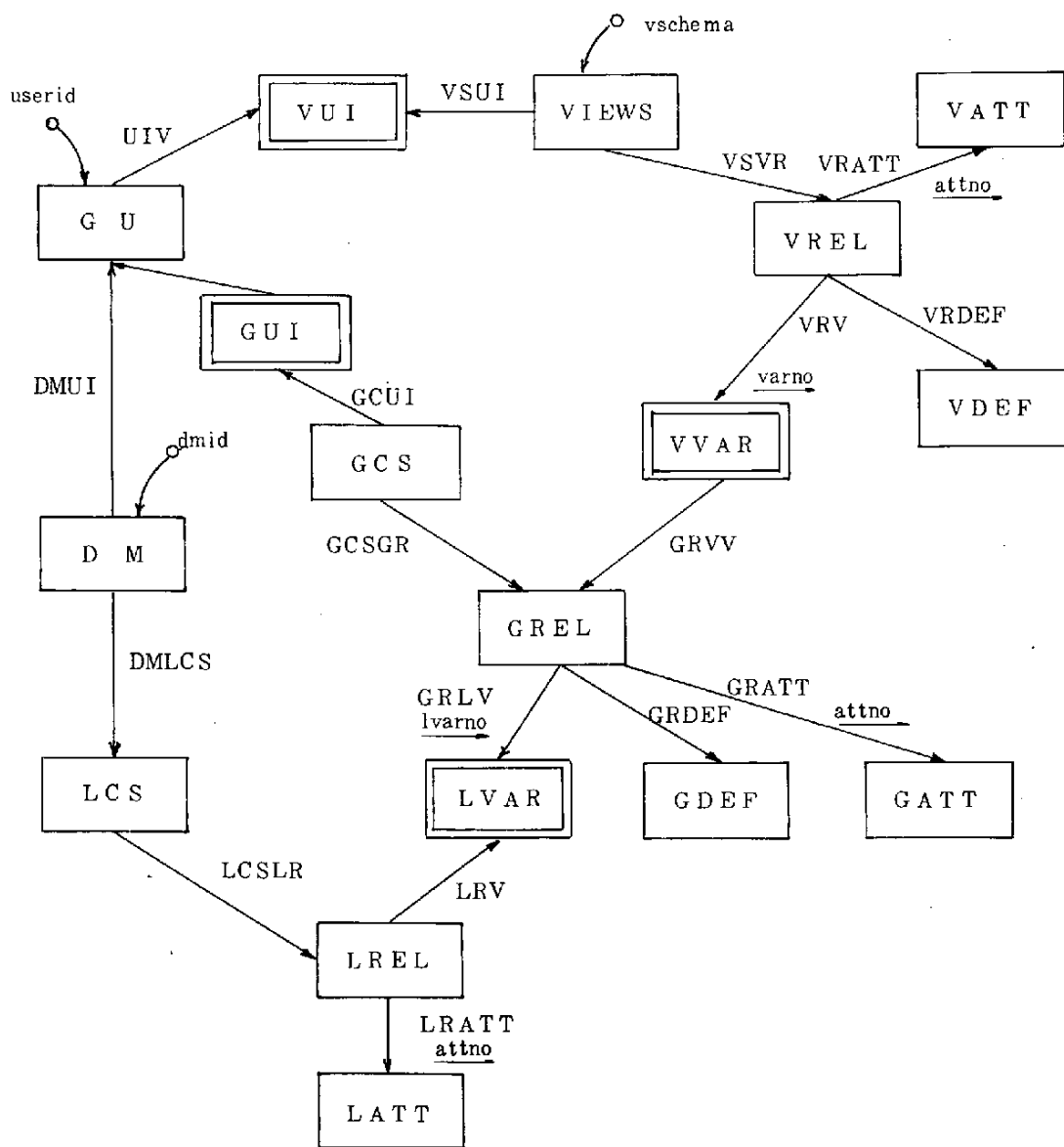


図 1.4 DIのCODASYLスキーマ

System Storage (D1)のスキーマを以下に示す。
スキーマ名は、DDDBSである。

説 明	D D L
スキーマ名	<pre> * * * ***** * * SCHEMA COMMAND * * ***** * * ***** * * SCHEMA ENTRY * * ***** * SCHEMA NAME IS DDDBS FOR DB; AUTHOR IS M-SUZUKI; DATE-WRITTEN 1982-07-01. REMARKS DATABASE FOR SYSTEM STORAGE. (HI,UI,VI,DMLI,RWS). </pre>
レコード型名	<pre> * * ***** * * RECORD ENTRY * * ***** * RECORD NAME IS GCS; LOCATION MODE IS RANDOM USING GCSNAME DUPLICATES ARE NOT ALLOWED. </pre>
GCS	<pre> * 02 GCSNAME ; PIC IS X(16). 02 RELNO ; PIC IS 9(02). 02 DEGREE ; PIC IS 9(02). 02 GAID ; PIC IS X(07). 02 CDATE ; PIC IS 9(02). * * *--- GCS ---* </pre>
GREL	<pre> * RECORD NAME IS GREL. * 02 RELNAME ; PIC IS X(16). 02 DEGREE ; PIC IS 9(02). 02 DEFSTND ; PIC IS 9(02). * * *--- GREL ---* </pre>
GDEF	<pre> * RECORD NAME IS GDEF. * 02 DEFST ; PIC IS X(12B). * * *--- GDEF ---* </pre>
GATT	<pre> * RECORD NAME IS GATT. * 02 ATTNO ; PIC IS 9(2). 02 ATTNAME ; PIC IS X(16). 02 OTYPE ; PIC IS X(01). 02 LENGTH ; PIC IS 9(04). 02 DOMAIN ; PIC IS X(16). 02 ATYPE ; PIC IS 9(01). * * *--- GATT ---* </pre>
DM	<pre> * RECORD NAME IS DM; LOCATION MODE IS RANDOM USING DMID DUPLICATES ARE NOT ALLOWED. </pre>
GU	<pre> * 02 DMID ; PIC IS 9(08). * * *--- GU ---* RECORD NAME IS GU; LOCATION MODE IS RANDOM USING USERID DUPLICATES ARE NOT ALLOWED. * 02 USERID ; PIC IS X(07). 02 PASSWD ; PIC IS X(05). 02 FNAME ; PIC IS X(20). 02 LNAME ; PIC IS X(20). 02 LGIN ; PIC IS 9(02). 02 ACCLUG ; PIC IS 9(04). 02 ACCGU ; PIC IS 9(04). 02 RUNF ; PIC IS 9(02). </pre>

説明	D	D	L
LCS	*--- LCS ---*		
	* RECORD NAME IS LCS.		
	02	LCSNAME	; PIC IS X(16).
	02	DEGREE	; PIC IS 9(02).
	02	LAID	; PIC IS X(07).
	02	RELNO	; PIC IS 9(02).
LVAR	*--- LVAR ---*		
	* RECORD NAME IS LVAR.		
	02	LVARNO	; PIC IS 9(02).
VUI	*--- VUI ---*		
	* RECORD NAME IS VUI.		
	02	RETFLG	; PIC IS 9(01).
	02	APPFLG	; PIC IS 9(01).
	02	REPFLG	; PIC IS 9(01).
	02	DELFLG	; PIC IS 9(01).
GUI	*--- GUI ---*		
	* RECORD NAME IS GUI.		
	02	RETFLG	; PIC IS 9(01).
	02	APPFLG	; PIC IS 9(01).
	02	REPFLG	; PIC IS 9(01).
	02	DELFLG	; PIC IS 9(01).
LREL	*--- LREL ---*		
	* RECORD NAME IS LREL.		
	02	RELNO	; PIC IS 9(02).
	02	RELNAME	; PIC IS X(16).
	02	RTYPE	; PIC IS X(01).
	02	DEGREE	; PIC IS 9(02).
	02	CARD	; PIC IS 9(08).
	02	WIDTH	; PIC IS 9(04).
VIEWS	*--- VIEWS ---*		
	* RECORD NAME IS VIEWS.		
	02	VSCHEMA	; PIC IS X(16).
	02	DEGREE	; PIC IS 9(02).
	02	VAID	; PIC IS X(07).
	02	CDATE	; PIC IS 9(06).
VREL	*--- VREL ---*		
	* RECORD NAME IS VREL.		
	02	VRELNAME	; PIC IS X(16).
	02	DEGREE	; PIC IS 9(02).
	02	DEFSTNO	; PIC IS 9(02).
VVAR	*--- VVAR ---*		
	* RECORD NAME IS VVAR.		
	02	VARNO	; PIC IS 9(02).
VDEF	*--- VDEF ---*		
	* RECORD NAME IS VDEF.		
	02	DEFST	; PIC IS X(80).
VATT	*--- VATT ---*		
	* RECORD NAME IS VATT.		
	02	ATTNO	; PIC IS 9(02).
	02	ATTNAME	; PIC IS X(16).
	02	DTYPE	; PIC IS X(01).
	02	LENGTH	; PIC IS 9(04).
	02	ATYPE	; PIC IS 9(01).
	02	DOMAIN	; PIC IS X(16).
LATT	*--- LATT ---*		

説 明	D	D	L
LATT	<pre> * RECORD NAME IS LATT. * 02 ATTNO ; PIC IS 9(02). 02 ATTNAME ; PIC IS X(16). 02 DTYPE ; PIC IS X(01). 02 LENGTH ; PIC IS 9(04). 02 DOMAIN ; PIC IS X(16). 02 SELECTIVITY ; PIC IS 9(07). 02 ROLE ; PIC IS 9(02). * </pre>		
セット型名	<pre> * * ***** * SET ENTRY * * ***** * </pre>		
GCSGR	<pre> * SET NAME IS GCSGR; OWNER IS GCS; MEMBER IS GREL LINKED TO OWNER PRIOR; INSERTION IS LAST. * </pre>		
GCUI	<pre> * * --- GCUI ---* SET NAME IS GCUI; OWNER IS GCS; MEMBER IS GUI LINKED TO OWNER PRIOR; INSERTION IS LAST. * </pre>		
GRDEF	<pre> * * --- GRDEF ---* SET NAME IS GRDEF; OWNER IS GREL; MEMBER IS GDEF LINKED TO OWNER PRIOR; INSERTION IS LAST. * </pre>		
GRATT	<pre> * * --- GRATT ---* SET NAME IS GRATT; OWNER IS GREL; MEMBER IS GATT LINKED TO OWNER PRIOR; INSERTION IS SORTED USING ATTNO DUPLICATES ARE LAST. * </pre>		
GRLV	<pre> * * --- GRLV ---* SET NAME IS GRLV; OWNER IS GREL; MEMBER IS LVAR LINKED TO OWNER PRIOR; INSERTION IS SORTED USING LVARNO DUPLICATES ARE LAST. * </pre>		
GRVV	<pre> * * --- GRVV ---* SET NAME IS GRVV; OWNER IS GREL; MEMBER IS VVAR LINKED TO OWNER PRIOR; INSERTION IS LAST. * </pre>		
LRV	<pre> * * --- LRV ---* SET NAME IS LRV; OWNER IS LREL; MEMBER IS LVAR LINKED TO OWNER PRIOR; INSERTION IS LAST. * </pre>		
DMUI	<pre> * * --- DMUI ---* SET NAME IS DMUI; OWNER IS DM; MEMBER IS GU LINKED TO OWNER PRIOR; INSERTION IS LAST. * </pre>		
DMLCS	<pre> * * --- DMLCS ---* SET NAME IS DMLCS; OWNER IS DM; MEMBER IS LCS LINKED TO OWNER PRIOR; </pre>		

説 明	D	D	L
	INSERTION IS LAST.		
UIV	* * SET NAME IS UIV; OWNER IS GU; MEMBER IS VUI LINKED TO OWNER PRIOR; INSERTION IS LAST.		*--- UIV ---*
UIG	* * SET NAME IS UIG; OWNER IS GU; MEMBER IS GUI LINKED TO OWNER PRIOR; INSERTION IS LAST.		*--- UIG ---*
LCSLR	* * SET NAME IS LCSLR; OWNER IS LCS; MEMBER IS LREL LINKED TO OWNER PRIOR; INSERTION IS LAST.		*--- LCSLR ---*
LRATT	* * SET NAME IS LRATT; OWNER IS LREL; MEMBER IS LATT LINKED TO OWNER PRIOR; INSERTION IS SORTED USING ATTNO DUPLICATES ARE LAST.		*--- LRATT ---*
VSUI	* * SET NAME IS VSUI; OWNER IS VIEWS; MEMBER IS VUI LINKED TO OWNER PRIOR; INSERTION IS LAST.		*--- VSUI ---*
VSVR	* * SET NAME IS VSVR; OWNER IS VIEWS; MEMBER IS VREL LINKED TO OWNER PRIOR; INSERTION IS LAST.		*--- VSVR ---*
VRATT	* * SET NAME IS VRATT; OWNER IS VREL; MEMBER IS VATT LINKED TO OWNER PRIOR; INSERTION IS SORTED USING ATTNO DUPLICATES ARE LAST.		*--- VRATT ---*
VRDEF	* * SET NAME IS VRDEF; OWNER IS VREL; MEMBER IS VDEF LINKED TO OWNER PRIOR; INSERTION IS LAST.		*--- VRDEF ---*
VRV	* * SET NAME IS VRV; OWNER IS VREL; MEMBER IS VVAR LINKED TO OWNER PRIOR; INSERTION IS SORTED USING VARNO DUPLICATES ARE LAST.		*--- VRV ---*
レンジ名	* * * *		***** * RANGE ENTRY * *****
RANGCCS	RANGE NAME IS RANGCCS; RECORD NAME IS GCS; PRIME IS 1.		

説 明	D	D	L
RANGDM	* RANGE NAME IS RANGDM; RECORD NAME IS DM; PRIME IS 1.		
RANGGU	* RANGE NAME IS RANGGU; RECORD NAME IS GU; PRIME IS 1.		
RANGGDF	* RANGE NAME IS RANGGDF; RECORD NAME IS GREL, GDEF, GATT; PRIME IS 60; TYPE IS CYCLE.		
RANGVG	* RANGE NAME IS RANGVG; RECORD NAME IS GUI; PRIME IS 2; TYPE IS CYCLE.		
RAUGUG	* RANGE NAME IS RANGUG; RECORD NAME IS VUI; PRIME IS 2; TYPE IS CYCLE.		
RANGLCS	* RANGE NAME IS RANGLCS; RECORD NAME IS LCS, LREL, LAIT, LVAR; PRIME IS 80; TYPE IS CYCLE.		
RANGVV	* RANGE NAME IS RANGVV; RECORD NAME IS VIEWS, VREL; PRIME IS 2; TYPE IS CYCLE.		
RANGVAT	* RANGE NAME IS RANGVAT; RECORD NAME IS VATT; PRIME IS 30; TYPE IS CYCLE.		
RANGGVD	* RANGE NAME IS RANGGVD; RECORD NAME IS VDEF; PRIME IS 10; TYPE IS CYCLE.		
RANGVVR	* RANGE NAME IS RANGVVR; RECORD NAME IS VVAR; PRIME IS 3; TYPE IS CYCLE.		
データセット名	* * * ***** * DATASET ENTRY * *****		
LRSSDB1	* DATASET NAME IS LRSSDB1; PAGE CONTAINS 2000. * VOLUME IS USER03; UNIT NAME F493; LOCATE RANGCCS, RANGDM, RANGGU, RANGGDF, RANGVG, RANGLCS, RANGGVD, RANGVV, RANGVAT, RANGUG, RANGVVR. * VOLUME IS WORK01;		

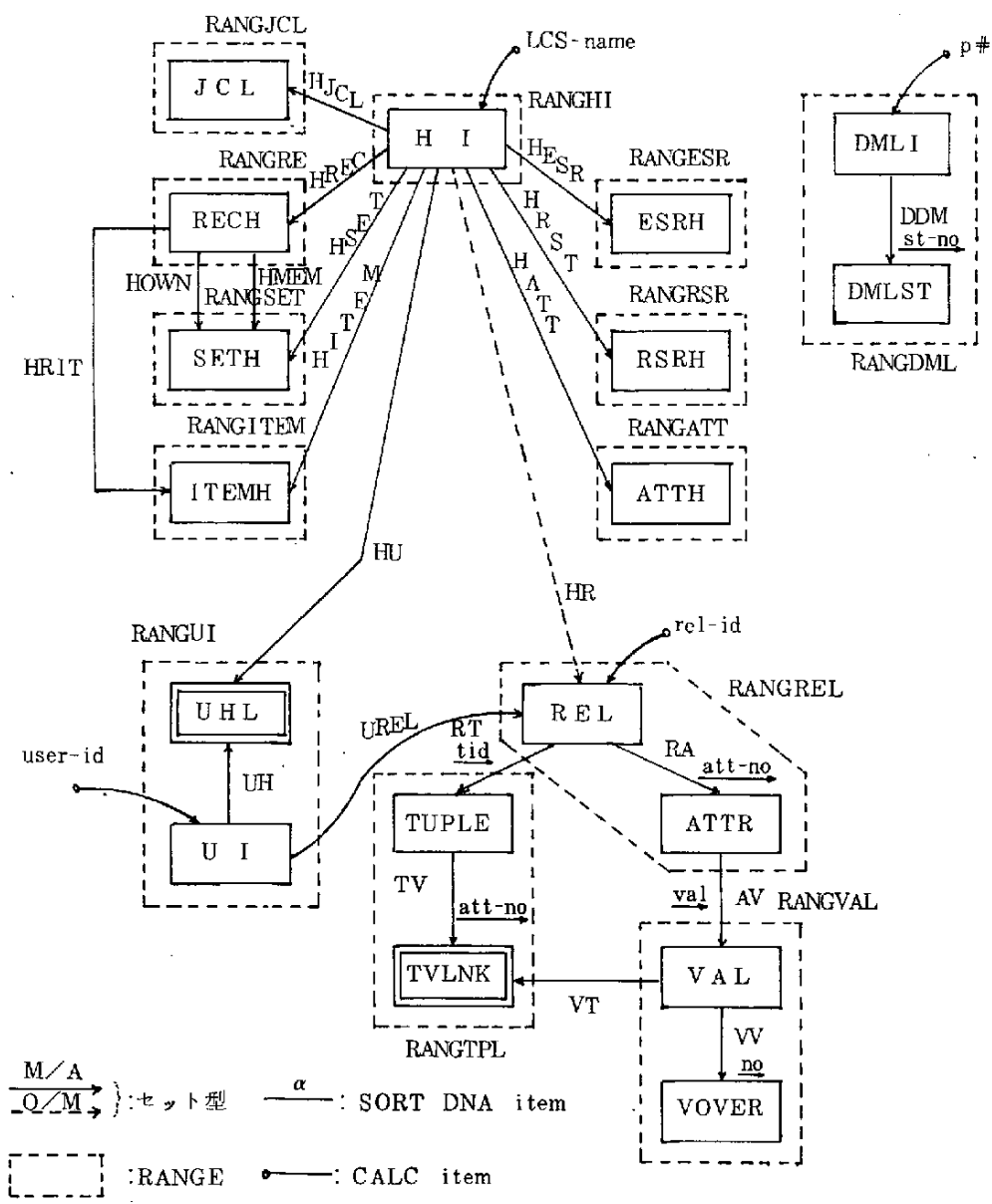


図 1.5 SS (System storage) のスキーマ(1)

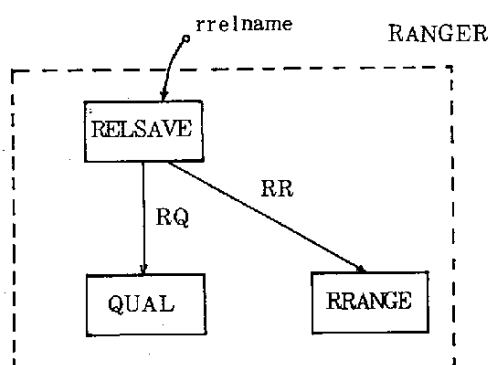
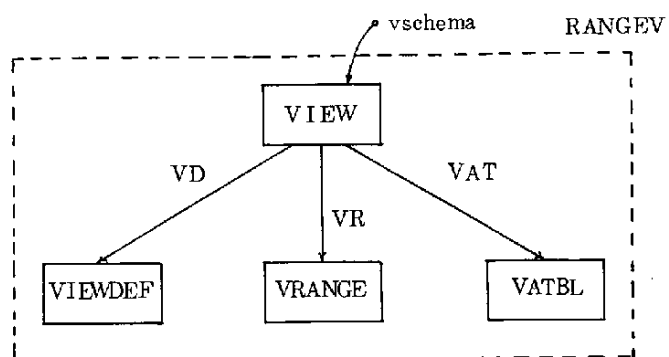


図 1.6 SS(System Storage)のスキーマ(2)

System Storage (HI, RWS)のスキーマを以下に示す。
スキーマ名は、LRSSDBである。

説 明	D	D	L
スキーマ名 LRSSDB	***** * SCHEMA COMMAND * *****		
	***** * SCHEMA ENTRY * *****		
レコード型 H I	SCHEMA NAME IS LRSSDB FOR DB: AUTHOR IS M-SUZUKI; DATE-WRITTEN 1982-07-01. REMARKS DATABASE FOR SYSTEM STORAGE. (HI,UL,VI,DMLI,RWS).		
	***** * RECORD ENTRY * *****		
E S R H	RECORD NAME IS HI; LOCATION MODE IS RANDOM USING LCSNAME DUPLICATES ARE NOT ALLOWED.		

R S R H	02 LCSNAME : PIC IS X(16). 02 LISNAME : PIC IS X(16). 02 HOSTID : PIC IS X(16). 02 DBNAME : PIC IS X(16). 02 LANAME : PIC IS X(20).		

A T T H	RECORD NAME IS ESRH.		

	02 ESREL : PIC IS X(16). 02 DEGREE : PIC IS 9(03). 02 TARGNAME : PIC IS X(16). 02 TARGTYPE : PIC IS 9(01).		

	RECORD NAME IS RSRH.		

	02 RSREL : PIC IS X(16). 02 RSTYPE : PIC IS 9(01). 02 ESNO : PIC IS 9(03). 02 ESREL : PIC IS X(16). 02 TARGNAME : PIC IS X(16). 02 TARGTYPE : PIC IS 9(01). 02 DEGREE : PIC IS 9(03). 02 ROLE : PIC IS X(16).		

	RECORD NAME IS ATTH.		

	02 RELNAME : PIC IS X(16). 02 RELTYPE : PIC IS 9(01). 02 ATTNO : PIC IS 9(03). 02 ATTNAME : PIC IS X(16). 02 ESNO : PIC IS 9(03). 02 DOMAIN : PIC IS X(16). 02 TYPED : PIC IS X(01). 02 LENGTH : PIC IS 9(05). 02 TARGNO : PIC IS 9(03). 02 TARGNAME : PIC IS X(16). 02 ROLE : PIC IS 9(02). 02 MARK : PIC IS 9(01).		

説明	D D L			
RECH	* * RECORD NAME IS RECH. *--- REC ---*			
	02	RECTYPE	: PIC IS	X(16).
	02	RECAUDE	: PIC IS	9(01).
	02	DEGREE	: PIC IS	9(03).
	02	CARDINALITY	: PIC IS	9(07).
	02	LUCNODE	: PIC IS	9(01).
	02	AREANAME	: PIC IS	X(16).
	02	MARK	: PIC IS	9(01).
	02	WGRK	: PIC IS	9(02).
	* * RECORD NAME IS SETH. *--- SETH ---*			
SETH	02	SETTYPE	: PIC IS	X(16).
	02	UREC	: PIC IS	X(16).
	02	MREC	: PIC IS	X(16).
	02	CONNECTIVITY	: PIC IS	9(05)V9(04).
	02	MAXCONNECT	: PIC IS	9(05)V9(04).
	02	MINCONNECT	: PIC IS	9(05)V9(04).
	02	MCCONNECT	: PIC IS	9(05)V9(04).
	02	SORTTYPE	: PIC IS	9(01).
	02	SORTATTN	: PIC IS	9(03).
	02	UNIQUEATT	: PIC IS	9(03).
	02	LINKMODE	: PIC IS	9(01).
	02	INSERTIONS	: PIC IS	9(01).
	02	RETENSIONS	: PIC IS	9(01).
	02	STRUCTONE	: PIC IS	9(03).
	02	STRUCTMEM	: PIC IS	9(03).
	02	MARK	: PIC IS	9(01).
	* * RECORD NAME IS ITEMH. *--- ITEM ---*			
	02	RECTYPE	: PIC IS	X(16).
ITEMH	02	ATTN	: PIC IS	9(03).
	02	ATTNAME	: PIC IS	X(16).
	02	NULNODE	: PIC IS	9(01).
	02	INDEXMODE	: PIC IS	9(01).
	02	UNIQUENESS	: PIC IS	9(01).
	02	SELECTIVITY	: PIC IS	9(07).
	02	TYPE1	: PIC IS	X(01).
	02	LENGTH1	: PIC IS	9(05).
	02	DOMAIN	: PIC IS	X(16).
	02	MARK	: PIC IS	9(01).
JCL	* * RECORD NAME IS JCL. *--- JCL ---*			
	02	JCLST	: PIC IS	X(80).
	02	STNU	: PIC IS	9(03).
UI	* * RECORD NAME IS UI; *--- UI ---*			
	LOCATION MODE IS RANDOM USING USERID DUPLICATES ARE NOT ALLOWED.			
	02	USERID	: PIC IS	X(07).
	02	FNAME	: PIC IS	X(20).
	02	LNAME	: PIC IS	X(20).
	02	PASSWORD	: PIC IS	X(05).
	02	LGDN	: PIC IS	9(02).
	02	ACCLCG	: PIC IS	9(04).
	02	ACGSD	: PIC IS	9(04).
	02	RUNF	: PIC IS	9(02).

説 明	D D L
UHL	<pre> * * *---- UHL ----* RECORD NAME IS UHL. * </pre>
DML I	<pre> * * *---- DML I ----* RECORD NAME IS DML I; LOCATION MODE IS RANDOM USING PN DUPLICATES ARE NOT ALLOWED. * 02 PN ; PIC IS X(08). 02 DN ; PIC IS 9(04). 02 SDN ; PIC IS 9(04). * </pre>
DMLST	<pre> * * *---- DMLST ---* RECORD NAME IS DMLST. * 02 STNO ; PIC IS 9(04). 02 DMLSTM ; PIC IS X(60). * </pre>
REL	<pre> * * *---- REL ----* RECORD NAME IS REL; LOCATION MODE IS RANDOM USING RELID DUPLICATES ARE NOT ALLOWED. * 02 RELID ; PIC IS 9(05). 02 RELNAME ; PIC IS X(16). 02 DEGREE ; PIC IS 9(04). 02 CARD ; PIC IS 9(07). 02 WIDTH ; PIC IS 9(09). 02 TPRMODE ; PIC IS 9(03). 02 FLAG ; PIC IS 9(03). 02 EXIST ; PIC IS 9(03). * </pre>
ATTR	<pre> * * *---- ATTR ----* RECORD NAME IS ATTR. * 02 ATTRID ; PIC IS 9(04). 02 ATTRNAME ; PIC IS X(16). 02 DTYPE ; PIC IS X(01). 02 DLNG ; PIC IS 9(04). 02 CARD ; PIC IS 9(09). 02 ATYPE ; PIC IS 9(03). 02 DOMAIN ; PIC IS X(16). * </pre>
VAL	<pre> * * *---- VAL ----* RECORD NAME IS VAL. * 02 DVAL ; PIC IS X(04). 02 CFLAG ; PIC IS 9(01). * </pre>
VOVER	<pre> * * *---- VOVER ----* RECORD NAME IS VOVER. * 02 DVAL ; PIC IS X(50). 02 BNO ; PIC IS 9(04). * </pre>
TUPLE	<pre> * * *---- TUPLE ----* RECORD NAME IS TUPLE. * 02 TID ; PIC IS X(27). 02 FLAG ; PIC IS 9(05). * </pre>
TVLNK	<pre> * * *---- TVLNK ----* RECORD NAME IS TVLNK. * 02 ATTNO ; PIC IS 9(04). * </pre>

説 明	D D L
VIEW	<pre> * *----- FOR VIEW RELATION ----- RECORD NAME IS VIEW; LOCATION MODE IS RANDOM USING VSCHEMA DUPLICATES ARE NOT ALLOWED. 02 VSCHEMA : PIC X(16).. </pre>
VIEWDEF	<pre> * RECORD NAME IS VIEWDEF. 02 VREL : PIC X(16). 02 STATEMENT : PIC X(80). </pre>
VATBL	<pre> * RECORD NAME IS VATBL. 02 VREL : PIC X(16). 02 ATTNO : PIC 9(3). 02 ATTNAME : PIC X(16). 02 RULE : PIC 9(1). 02 TYPEV : PIC X(1). 02 LENGTH : PIC 9(5). 02 DOMAIN : PIC X(16). </pre>
VRANGE	<pre> * RECORD NAME IS VRANGE. 02 GVAR : PIC X(04). 02 GREL : PIC X(16). </pre>
RELSAVE	<pre> * RECORD NAME IS RELSAVE; LOCATION MODE IS RANDOM USING RRELNAME DUPLICATES ARE NOT ALLOWED. 02 RRELNAME : PIC X(16). </pre>
QUAL	<pre> * RECORD NAME IS QUAL. 02 QUALST : PIC X(80). </pre>
RRANGE	<pre> * RECORD NAME IS RRANGE. 02 RRANGST : PIC X(80). </pre>
セット型	<pre> * * * ***** * SET ENTRY * ***** </pre>
HESR	<pre> * *----- HESR ----- SET NAME IS HESR; OWNER IS H1; MEMBER IS LSRH LINKED TO OWNER PRIOR; INSERTION IS LAST. </pre>
HRSR	<pre> * *----- HRSR ----- SET NAME IS HRSR; OWNER IS H1; MEMBER IS RSRH LINKED TO OWNER PRIOR; INSERTION IS LAST. </pre>
HATT	<pre> * *----- HATT ----- SET NAME IS HATT; OWNER IS H1; MEMBER IS ATTH LINKED TO OWNER PRIOR; INSERTION IS LAST. </pre>
HREC	<pre> * *----- HREC ----- SET NAME IS HREC; OWNER IS H1; MEMBER IS RECH LINKED TO OWNER PRIOR; INSERTION IS LAST. </pre>
HSET	<pre> * *----- HSET ----- SET NAME IS HSET; OWNER IS H1; MEMBER IS SETH LINKED TO OWNER PRIOR; </pre>

説 明	D D L
HITEM	<pre> * INSERTION IS LAST. * * *---- HITEM ----* SET NAME IS HITEM; OWNER IS HIZ; MEMBER IS ITEMH LINKED TO OWNER PRIOR; INSERTION IS LAST. </pre>
HJCL	<pre> * *---- HJCL ----* SET NAME IS HJCL; OWNER IS HIZ; MEMBER IS JCL LINKED TO OWNER PRIOR; INSERTION IS SORTED USING STNO DUPLICATES ARE LAST. </pre>
HRIT	<pre> * *---- HRIT ----* SET NAME IS HRIT; OWNER IS RECH; MEMBER IS ITEMH LINKED TO OWNER PRIOR; INSERTION IS SORTED USING ATTND DUPLICATES ARE LAST. </pre>
HOWN	<pre> * *----- HOWN -----* SET NAME IS HOWN; OWNER IS RECH; MEMBER IS SETH LINKED TO OWNER; INSERTION IS LAST. </pre>
HMEM	<pre> * *----- HOWN -----* SET NAME IS HMEM; OWNER IS RECH; MEMBER IS SETH LINKED TO OWNER; INSERTION IS LAST. </pre>
HU	<pre> * *----- HU -----* SET NAME IS HU; OWNER IS HIZ; MEMBER IS UHL LINKED TO OWNER; INSERTION IS LAST. </pre>
UH	<pre> * *----- UH -----* SET NAME IS UH; OWNER IS UI; MEMBER IS UHL LINKED TO OWNER; INSERTION IS LAST. </pre>
UREL	<pre> * *---- UREL ----* SET NAME IS UREL; OWNER IS UI; MEMBER IS REL LINKED TO OWNER PRIOR; INSERTION IS LAST. </pre>
HR	<pre> * *---- HR ----* SET NAME IS HR; OWNER IS HIZ; MEMBER IS REL LINKED TO OWNER PRIOR; INSERTION IS LAST. </pre>
RA	<pre> * *----- RA -----* SET NAME IS RA; OWNER IS REL; MEMBER IS ATTR LINKED TO OWNER PRIOR; INSERTION IS SORTED USING ATTND DUPLICATES ARE NOT. </pre>

説 明	D D L
AV	<pre> * * *----- AV -----* SET NAME IS AV; OWNER IS ATTR; MEMBER IS VAL LINKED TO OWNER PRIOR; INSERTION IS SORTED USING DVAL DUPLICATES ARE LAST. </pre>
VV	<pre> * * *----- VV -----* SET NAME IS VV; OWNER IS VAL; MEMBER IS VOVER LINKED TO OWNER PRIOR; INSERTION IS SORTED USING AND DUPLICATES ARE LAST. </pre>
VT	<pre> * * *----- VT -----* SET NAME IS VT; OWNER IS VAL; MEMBER IS TVLNK LINKED TO OWNER; INSERTION IS LAST. </pre>
TV	<pre> * * *----- TV -----* SET NAME IS TV; OWNER IS TUPLE; MEMBER IS TVLNK LINKED TO OWNER; INSERTION IS SORTED USING ATTNO DUPLICATES ARE NOT. </pre>
RT	<pre> * * *----- RT -----* SET NAME IS RT; OWNER IS REL; MEMBER IS TUPLE LINKED TO OWNER PRIOR; INSERTION IS SORTED USING TID DUPLICATES ARE NOT. </pre>
DOM	<pre> * * *----- DOM -----* SET NAME IS DOM; OWNER IS DML; MEMBER IS DMLST LINKED TO OWNER PRIOR; INSERTION IS SORTED USING LGNO DUPLICATES ARE NOT. </pre>
VD	<pre> * *----- SET FOR VIEW -----* * SET NAME IS VD; OWNER IS VIEW; MEMBER IS VIEWDEF LINKED TO OWNER; INSERTION IS LAST. </pre>
VAT	<pre> * SET NAME IS VAT; OWNER IS VIEW; MEMBER IS VAIBL; INSERTION IS LAST. </pre>
VR	<pre> * SET NAME IS VR; OWNER IS VIEW; MEMBER IS VRANGE; INSERTION IS LAST. </pre>
RQ	<pre> * SET NAME IS RQ; OWNER IS RELSAVE; MEMBER IS QUAL; INSERTION IS LAST. </pre>
RR	<pre> * SET NAME IS RR; OWNER IS RELSAVE; MEMBER IS RRANGE; INSERTION IS LAST. </pre>

説 明	D D L
レンジ名	<pre> * * * * ***** * RANGE ENTRY * ***** </pre>
RANGHM	<pre> * RANGE NAME IS RANGHM; RECORD NAME IS HI; PRIME IS 3. </pre>
RANGJCL	<pre> * RANGE NAME IS RANGJCL; RECORD NAME IS JCL; PRIME IS 25. </pre>
RANGESR	<pre> * RANGE NAME IS RANGESR; RECORD NAME IS ESRH; PRIME IS 10. </pre>
RANGRSR	<pre> * RANGE NAME IS RANGRSR; RECORD NAME IS RSRH; PRIME IS 20. </pre>
RANGATT	<pre> * RANGE NAME IS RANGATT; RECORD NAME IS ATTH; PRIME IS 50. </pre>
RANGREC	<pre> * RANGE NAME IS RANGREC; RECORD NAME IS RECH; PRIME IS 15. </pre>
RANGSET	<pre> * RANGE NAME IS RANGSET; RECORD NAME IS SETH; PRIME IS 30. </pre>
RANGITEM	<pre> * RANGE NAME IS RANGITEM; RECORD NAME IS ITEMH; PRIME IS 50. </pre>
RANGDML	<pre> * RANGE NAME IS RANGDML; RECORD NAME IS DML1, DMLST; PRIME IS 100; LOGICAL PAGE IS 5. </pre>
RANGUI	<pre> * RANGE NAME IS RANGUI; RECORD NAME IS UI, UHL; PRIME IS 2. </pre>
RANGREL	<pre> * RANGE NAME IS RANGREL; RECORD NAME IS REL, ATTH; PRIME IS 100. </pre>
RANGVAL	<pre> * RANGE NAME IS RANGVAL; RECORD NAME IS VAL, VUVER; PRIME IS 70; OVERFLOW IS 10; LOGICAL PAGE IS 5. </pre>
RANGTDL	<pre> * RANGE NAME IS RANGTDL; RECORD NAME IS TUPLE, TVLNK; PRIME IS 35; OVERFLOW IS 5; LOGICAL PAGE IS 5. </pre>

説 明	D	D	L
	<pre> * * *----- RANGE FOR VIEW ----- * RANGE NAME IS RANGEV; RECORD NAME IS VIEW, VIEWDEF, VATSL, VRANGE; PRIME IS 30; OVERFLOW IS 10; LOGICAL PAGE IS 5; RATE IS 30. </pre>		
RANGEV			
	<pre> * RANGE NAME IS RANGER; RECORD NAME IS RELSAVE, RUAL, RRANGE; PRIME IS 30; OVERFLOW IS 10; LOGICAL PAGE IS 5; RATE IS 30. </pre>		
RANGER			
データセット名	<pre> * * *----- * DATASET ENTRY * *----- </pre>		
LRSSDB	<pre> * DATASET NAME IS LRSSDB; PAGE CONTAINS 2000. </pre>		
	<pre> * VOLUME IS USER05; UNIT NAME F493; LOCATE RANGH1, RANGESR, RANGRSR, RANGATT, RANGJCL, RANGREC, RANGSET, RANGITEM, RANGUI, RANGHEL, RANGVAL, RANGTPL, RANGDML, RANGEV, RANGER. </pre>		
	<pre> * END. </pre>		

2. ローカルデータベースプロセッサ (LDP-V3)

CODASYLデータベースシステム上の関係インタフェースシステムとしてのローカルデータベースプロセス(LDP-V3)は、以下の機能を有している。

- (1) CODASYLスキーマから関係スキーマと両スキーマの対応情報とを生成する〔スキーマ変換〕
- (2) 関係スキーマ（ローカル概念（LC）スキーマ（LCS））上のRQLによる関係問合せ、及び関係更新を、COBOL DMLに変換し、実行し、結果を関係として導出する〔問合せ変換・更新変換〕

2.1 LDP-V3のシステム構成

LDP-V3は、図2・1に示す様に次の4つのモジュールから成っている。

- (1) H I P (heterogeneity information processor)
- (2) Q T P (query translation processor)
- (3) H I (heterogeneity information)
- (4) R W S (relational working storage)

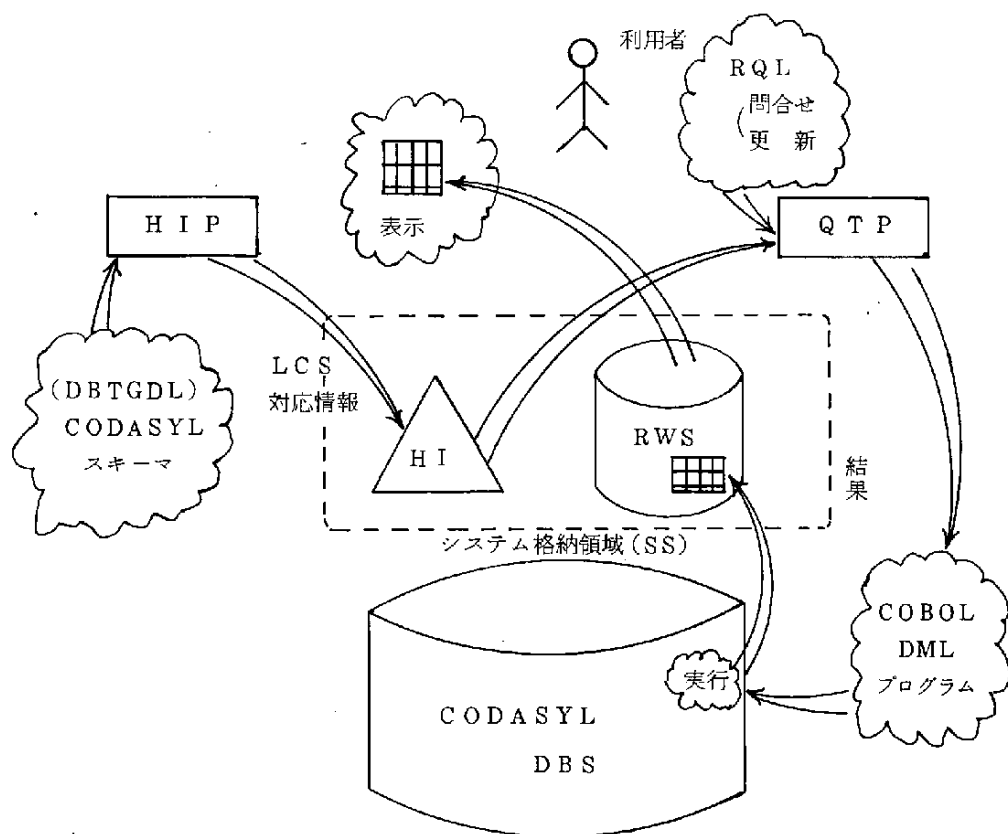


図 2.1 LDP-V3 のモジュール構成

HIP (heterogeneity information processor) は、DBTGDL で記述した CODASYL スキーマ (DBTGD) を入力として、LCS スキーマ及び LCS と CODASYL スキーマとの対応情報を生成する。この情報は異種性情報 HI (heterogeneity information) に格納される。HI は、他に、COBOL DML の情報 (DMLI) を有している。

QTP (query translation processor) は、LC スキーマに対する利用者の RQL 問合せ (≒ QUEL 問合せ) 及び RQL 更新を受けて、CODASYL スキーマ上の COBOL DML プログラムに変換し、CODASYL データベースシステム上で実行させて、解を関係として、関係作業領域 (RWS (relational working storage)) に格納する。RWS 内に導出された関係は、LDP-V3 によって、利用者に表示される。QTP は、RQL から、COBOL DML への変換時に、必要な変換情報を、HI から得る。従って、QTP は目標データベースとは独立につくられている。HI を他のデータベースシステム用に変えることによって、容易に、この DBS 用の関係インタフェースとなる。

2.2 HIP の実現

スキーマ変換を行う HI プロセッサ (HIP) では、DBTGDL で書かれた DBTGD (CODASYL スキーマ記述) 及び DMLD (DML 記述) を入力し、CODASYL スキーマ (LIS) から関係スキーマ (LCS) を生成する。スキーマ変換プロセッサの概要は、図 2.2 の様になっている。

- LCSSI LCS
- ESR
- RSR
- ATT

- DMLI-DML

LISG は、DBTGD と DMLD を入力して、LISSI の 4 つの関係 (LIS、REC、SET、ITEM) と DMLI の DML 関係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アクセスし、LISSI の関係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 つの関係 (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。

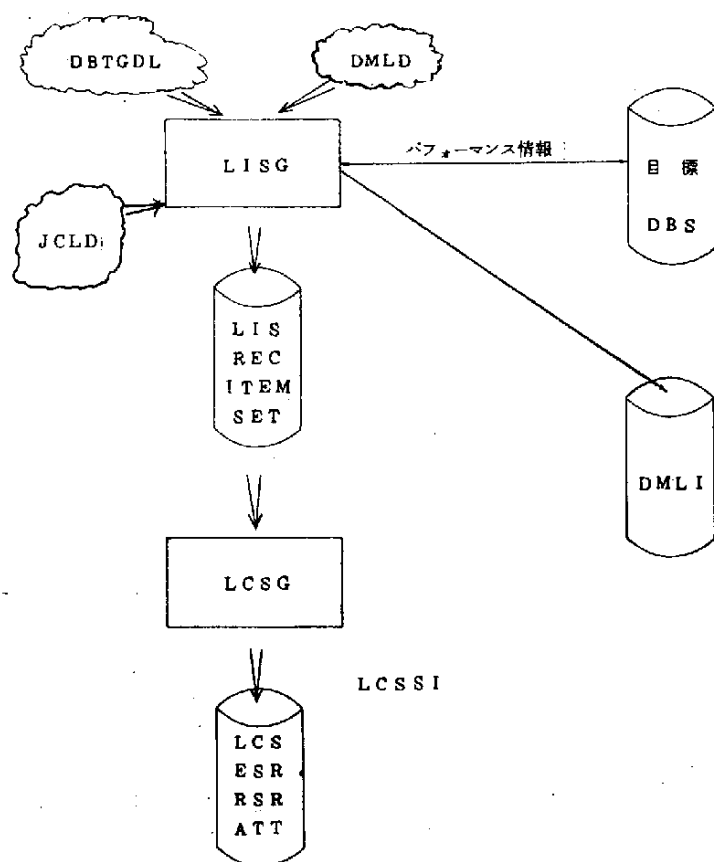


図2.2 HIPの処理概要

① レコード型

・項目の構成

各項目毎に項目名、タイプ、長さ、

NNA (null is not allowed) 又は NA (null is allowed)

・各項目についてのアクセスパス

CALC with DNA-CALC項目で重複されない項目。

CALC with DA-CALC項目で重複されている項目。

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。

・リンクの為の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 (M/A 又は O/M とする)

insertion AUTOMTIC/MANUAL

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。
 ソートモード —昇順又は降順

・構造制約 (structural constraint)

・set-selection

CODASYLスキーマ記述言語 (DBTGDL) の構文を図2.3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> ::= <fieldname> / <format>
                       [ / <access-mode> [ / <NNA-mode> ] ]

<access-mode> ::= <calc-mode> / <dna-mode>

<dna-mode> ::= DNA | DA *

<calc-mode> ::= CALC | NCALC *

<format> ::= <type> <length>

<NNA-mode> ::= NNA | NA *

<type> ::= C | D      (註) *は、省略時解釈を示す。
```

図2.3 DBTGDLの定義(1)

```
<set-type-description> ::= SET [ -TYPE ]
                           <set-type>
                           <owner-description>
                           <member-description>
                           [ <structural constraint> ]
```

```

< owner-description > ::= OWNER [ 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 | PRIOR
< structural-comstraint > ::= STRUCTURE ( < att-name-of-o-rec > )
                                < att-name-of-m-rec > )

```

図2・4 DBTGDLの定義(2)

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。

DMLDLの構文を図2・5に示す。

```

< DMLD > ::= < DML-block-def >
                { < DML-block-def > }
< DML-block-def > ::= < block-no > // < DML-statement >
                    { // < DML-statement > } ;
< block-no > ::= P < number >

```

図2・5 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、計算機、データベース、言語が変わっても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（関係スキーマ）の記述情報
- ・LISとLCSの対応情報
- ・DML情報
- ・JCL情報

また、これらの情報は次の4) 種類に分割され、各々関係モデルで、関係として表わされる。

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関係から成る。

- ・LIS関係
- ・REC関係
- ・SET関係
- ・ITEM関係

2) LCSSI

LCSSIは、関係スキーマ情報とLISとの対応情報を表わした次の様な関係から成る。

- ・LCS関係
- ・ESR関係
- ・RSR関係
- ・ATT関係

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関係を持つ。

DML言語が異なる場合、このDML関係を変更すればよい。

4) JCLI

JCL情報を有する。

a) LISSI

LISSIは、CODASYLスキーマを関係モデルで表わした3つの関係（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）を持つ関係（LIS）とから成る。各関係の構造は、表2・1、2・2、2・3、2・4の様になっている。

LIS関係は、各データベースシステムのLISスキーマとLIデータベースについての情報を保持している。ホスト計算機名、スキーマ名、サブスキーマ名は、DML生成時に参照される。

REC関係は、CODASYLスキーマのレコード型を関係で表わしたものである。レコード実現値数はアクセス木生成の評価関数によって参照される。又、配置モードやエリア名はDML生成で参照される。

SET関係は、CODASYLスキーマのセット型を関係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関係は、CODASYLスキーマのレコード型のデータ項目を関係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、UNIQUENESS、TYPE、LENGTH等はDML生成で参照される。

b) LCSSI

LCSSIは、関係スキーマ情報と、CODASYLモデル要素への対応情報を表わしている。LCSSIは、関係スキーマ情報を表わす3つの関係(RSR、ESR、ATT)と関係スキーマについての情報(アクセス言語、LISとの対応情報)をもつ関係(LCS)とから成る[表2.6、2.7、2.8、2.9]。

ESR関係は、CODASYLスキーマのレコード型に対応する関係である。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、関係の存在の確認の為に参照される。

RSR関係は、CODASYLスキーマのセット型やリンクレコード型に対応する関係である。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、関係の存在を確認する為に参照される。

ATT関係は、CODASYLスキーマのデータ項目に対応する関係である。データ項目との対応情報(target-no, target-name)を持ち、構造変換で参照される。又、LCS問合せの入力で、関係の属性の存在を確認する為に参照される。

LCS関係は、LIS関係との対応情報を保持しており、構造変換で使用される。

D. 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関係から成っている。DMLIの構造は、表2.5のようになっている。

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(付記4 DMLI生成を参照)とのマッチングで参照される。pflagは、DML-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。

E. HIP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スキーマを関係モデルで記述したLISSIとDMLIを生成する。次に、LISSIを入力して、CODASYLデータ構造を関係形式で表現したLCSSを生成する。

HIPのシステム構成は、図2・6の様になっている。

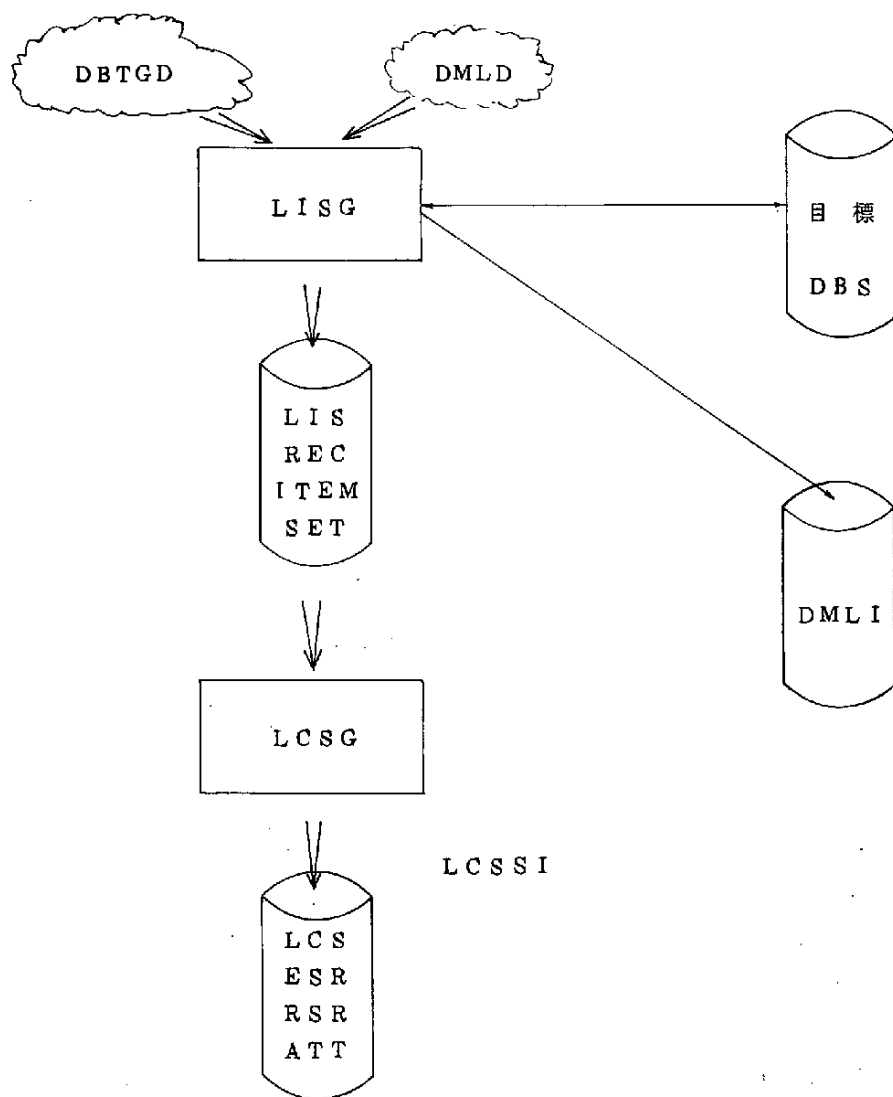


図2・6 HIPの処理概要

a) LISG

LISGは、DBTGDとDMLDを入力して、LISSIの4つの関係(LIS、REC、SET、ITEM)とDML関係を生成する。

また、LISSI内のパフォーマンス情報は、目標とするLIデータベースを実際にアクセ

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関係を生成する。

① DBTGDからLISSIの生成

ここでは、DBTGDの文字列を読み込み、LISSIの4つの関係を生成する。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。

[処理概要]

- 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関係に格納する。
- ハ) このレコード型をREC関係に格納する。
- ニ) 全てのRECORD文の処理が終わったら、SET文(SET……;)を読み込む。
- ホ) セット型をSET関係に格納する。
- ヘ) SCHEMA文(SCHMA……;)からLIS関係を生成する。 □

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関係のカーディナリティ、SET関係の結合度、最大、最少結合度、逆結合度(子から親への結合度)、ITEM関係の選択度を指す。

③ DMLDからDMLIの生成

block-noを全てのstatementに附加しながら、DML関係に出力する。

b) LISSIからLCSSIの生成

LISSIは、CODASYLモデル構造を関係形式で表現したものである。

LISSIを入力として、LCSSIとして4つの関係(LCS、RSR、ESR、ATT)を生成する。

次の様な処理で、LISSIからLCSSIを生成する。

[処理概要]

- 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関係(RSR)を生成する。

⇒ 辺に対して、T又はP-関係(RSR)を生成する。□

表 2 . 1 REC 関係の構成

属 性 名	属性番号	ROLE	TYPE	LENGTH	説 明
REC-TYPE	1	K	C	16	レコード型
REC-MODE	2	N	D	1	{ normal (= 0) link (= 1)
DEGREE	3	N	D	2	このレコード型のデータ項目の数
CARDINALITY	4	N	D	6	実現数
LOC-MODE	5	N	D	1	{ CALC (= 1) NONCALC (= 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	作業用

表2.2 SET 関係の構成

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	親レコード型名
m-rec-type	3	N	C	16	子レコード型名
connectivity	4	N	D	7	orec→mrec の平均connectivity
max-connect	5	N	D	7	orec→mrec の最大 "
min-connect	6	N	D	7	orec→mrec の最小 "
i-connect	7	N	D	7	mrec→orec の平均 "
sort-type	8	N	D	1	N(=0): not sorted A(=1): ascending D(=2): descending
sort-att#	9	N	D	2	ソートキーの属性番号
unique-att#	10	N	D	2	DNA指定のある属性番号
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)
retension	13	N	D	1	MANDATORY(=0) OPTIONAL(=1) FIXED(=2)

属 性 名	属性番号	ROLE	TYPE	LENGTH	説 明
structure- o-att #	14	N	D	2	structural constraint の owner att-no
structure- m-att #	15	N	D	2	" member att-no
mark	16	N	D	1	作業用

表 2・3 ITEM 関係の構成

属 性 名	属性番号	ROLE	TYPE	LENGTH	説 明
REC-TYPE	1	K	C	16	レコード型名
ATT-NO	2	K	D	2	データ項目番号
ATT-NAME	3	N	D	16	データ項目名
INDEX-MODE	4	N	D	1	{ CALC (= 1) NCALC (= 0)
UNIQUENESS	5	N	D	1	{ DNA (= 1) DA (= 0)
SELECTIVITY	6	N	D	6	選択度 × 1000 の値が入る。
TYPE	7	N	D	1	{ 'C' : 文字 'D' : 整数
LENGTH	8	N	D	6	バイト長
DOMAIN	9	N	D	16	定義域名
MARK	10	N	D	1	作業用 { OFF (= 0) ON (= 1)

表 2 . 4 L I S 関係の構成

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 (L I S の名前)
la-name	3	N	C	16	local administrator 名
host-computer	4	N	C	16	計算機名
os-name	5	N	C	16	O S 名
c-year	6	N	D	1	L I S の生成年
c-month	7	N	D	1	" 月
c-day	8	N	D	1	" 日
lis-model	9	N	C	4	'REL' = relational 'DBTG' = CODASYL DBTG 'IMS' = IMS
language	10	N	C	16	アクセス言語名 (e.g. COBOL DML)

表2・5 DML 関係の構成

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 = 予、%を含まない。 1 = " を含む。
statement-no	3	N	D	3	ブロック内の文番号
DML-statement	4	N	C	72	COBOL DMLの1文

表2・6 ESR 関係の構成

属 性 名	属性番号	ROLE	TYPE	LENGTH	説 明
ES-REL	1	K	C	16	E関係スキーム名
DEGREE	2	N	D	2	E関係スキームの属性の数
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)

表 2・7 RSR 関係の構成

属 性 名	属性番号	ROLE	TYPE	LENGTH	説 明
rs-rel	1	K	C	16	R関係スキーム名
rs-type	2	N	D	1	$\left\{ \begin{array}{l} T (=1) \text{ 型 (M/A)} \\ P (=2) \text{ 型 (O/M)} \\ G (=3) \text{ 型 (linkレコード型)} \end{array} \right.$
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	$\left\{ \begin{array}{l} \text{set} (=2) \\ \text{link-record} (=1) \end{array} \right.$
degree	7	N	D	1	属性数
role	8	N	C	16	$\left\{ \begin{array}{l} \text{rs-relがlink-record-typeの} \\ \text{とき関連するセット型名} \\ \text{rs-relがset-typeのとき} \\ \text{'owner' or 'member' } \end{array} \right.$

表 2. 8 A T T 関係の構成

属 性 名	属性番号	ROLE	TYPE	LENGTH	説 明
REL-NAME	1	K	C	16	E or R関係スキーム名
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	対応するレコード型名
ROLE	11	N	D	1	{ 1 : key 0 : non-key
mark	12	N	D	1	作業用

表2・9 LCS 関係の構成

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	ローカル管理者(DBA)名
lis-name	3	N	C	16	対応するLIS名
model	4	N	C	4	LCSモデル(relational)
language	5	N	C	16	LCSアクセス言語(RQL)
c-year	6	N	D	1	生成年
c-month	7	N	D	1	" 月
c-day	8	N	D	1	" 日

E. HIPの実働例

a) PRDBS

AIM上に定義されたPRDBSのスキーマを以下に示す。

説明	D D L
スキーマ名 PROJDBS	<pre> * * ***** * * SCHEMA COMMAND * * ***** * * ***** * * SCHEMA ENTRY * * ***** SCHEMA NAME IS PROJDBS FOR DB: AUTHOR IS M-SUZUKI; DATE-WRITTEN : 1980-08-18. REMARKS PROJECT DATABASE SYSTEM. </pre>
レコード型名 EMPLOYEE	<pre> * * ***** * * RECORD ENTRY * * ***** RECORD NAME IS EMPLOYEE. 02 END ; 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). </pre>
PROJECT	<pre> * 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-F ; PIC IS X(03). 02 SYEAR ; PIC IS 9(02). 02 EYEAR ; PIC IS 9(02). 02 BUDGET ; PIC IS 9(11). </pre>
REPORTR	<pre> * 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(203). 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 CUST ; PIC IS 9(09). 02 RTYPE ; PIC IS X. 02 RSTATE ; PIC IS X. </pre>

説 明	D D L
SPOUSOR	<pre> * RECORD NAME IS SPONSOR. * * NO CALC KEY * 02 SNAME ; PIC IS X(20). 02 STYPE ; PIC IS X(10). </pre>
SUBJECT	<pre> * RECORD NAME IS SUBJECT; LOCATION MODE IS RANDOM USING SUBJECTN DUPLICATES ARE NOT ALLOWED. 02 SUBJECTN ; PIC IS X(100). </pre>
PRODUCT	<pre> * 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). </pre>
ORGANIZA TIOR	<pre> * RECORD NAME IS ORGANIZATION. 02 ORG-NAME ; PIC IS X(50). 02 ORG-TYPE ; PIC IS X(10). * ***** * LINK RECORD TYPE * ***** </pre>
PROJ-EMP -LNK	<pre> * RECORD NAME IS PROJ-EMP-LNK. 02 STATUS ; 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). </pre>
EMP-REP- LNK	<pre> * RECORD NAME IS EMP-REP-LNK. 02 AUTH-NO ; PIC IS 9(02). </pre>
REP-SUBJ	<pre> * RECORD NAME IS REP-SUBJ. </pre>
PROJ-SUBJ	<pre> * RECORD NAME IS PROJ-SUBJ. </pre>
PROJ-RE	<pre> * RECORD NAME IS PROJ-RE. * * * ***** * SET ENTRY * * ***** * </pre>
セット型名 LEADER	<pre> * SET NAME IS LEADER; OWNER IS EMPLOYEE; MEMBER IS PROJECT LINKED TO OWNER PRIOR; INSERTION IS SORTED USING PNO DUPLICATES ARE NOT ALLOWED. </pre>
SPON-PROJ	<pre> * SET NAME IS SPON-PROJ; OWNER IS SPONSOR; MEMBER IS PROJECT LINKED TO OWNER PRIOR; INSERTION IS SORTED USING PNO DUPLICATES ARE NOT ALLOWED. </pre>
PROJ-PROD	<pre> * SET NAME IS PROJ-PROD; OWNER IS PROJECT; MEMBER IS PRODUCT LINKED TO OWNER PRIOR; INSERTION IS SORTED USING PD-NO DUPLICATES ARE NOT ALLOWED. </pre>

説 明	D	D	L
P-E	* SET NAME IS OWNER IS MEMBER IS INSERTION IS DUPLICATES ARE LAST.	P-E; PROJECT; PROJ-EMP-LNK SORTED USING YEAR	LINKED TO OWNER PRIOR; SYEAR
E-P	* SET NAME IS OWNER IS MEMBER IS INSERTION IS DUPLICATES ARE LAST.	E-P; EMPLOYEE; PROJ-EMP-LNK SORTED USING YEAR	LINKED TO OWNER PRIOR; SYEAR
P-S	* SET NAME IS OWNER IS MEMBER IS INSERTION IS	P-S; PROJECT; PROJ-SUBJ LAST.	LINKED TO OWNER PRIOR;
S-P	* SET NAME IS OWNER IS MEMBER IS INSERTION IS	S-P; SUBJECT; PROJ-SUBJ LAST.	LINKED TO OWNER PRIOR;
S-R	* SET NAME IS OWNER IS MEMBER IS INSERTION IS	S-R; SUBJECT; REP-SUBJ LAST.	LINKED TO OWNER PRIOR;
R-S	* SET NAME IS OWNER IS MEMBER IS INSERTION IS	R-S; REPORTR; REP-SUBJ LAST.	LINKED TO OWNER PRIOR;
R-E	* SET NAME IS OWNER IS MEMBER IS INSERTION IS DUPLICATES ARE NOT ALLOWED.	R-E; REPORTR; EMP-REP-LNK SORTED USING AUTH-NO	LINKED TO OWNER PRIOR;
E-R	* SET NAME IS OWNER IS MEMBER IS INSERTION IS DUPLICATES ARE LAST.	E-R; EMPLOYEE; EMP-REP-LNK SORTED USING AUTH-NO	LINKED TO OWNER PRIOR;
PRIOR -PROJ	* SET NAME IS OWNER IS MEMBER IS	PRIOR-PROJ; PROJECT; PROJ-RE	LINKED TO OWNER PRIOR.
POST-PROJ	* SET NAME IS OWNER IS MEMBER IS	POST-PROJ; PROJECT; PROJ-RE	LINKED TO OWNER PRIOR.
ORG-EMP	* SET NAME IS OWNER IS MEMBER IS INSERTION IS DUPLICATES ARE NOT ALLOWED.	ORG-EMP; ORGANIZATION; EMPLOYEE SORTED USING END	LINKED TO OWNER PRIOR;
SYS-ORG	* SET NAME IS OWNER IS MEMBER IS	SYS-ORG; SYSTEM; ORGANIZATION.	
SYS-SPON	* SET NAME IS OWNER IS MEMBER IS	SYS-SPON; SYSTEM; SPONSOR.	
SYS-EMP	* SET NAME IS OWNER IS MEMBER IS INSERTION IS DUPLICATES ARE NOT ALLOWED.	SYS-EMP; SYSTEM; EMPLOYEE; SORTED USING END	

説 明	D D L
データセット名 PROJDB	<pre> * * * DATASET NAME IS PROJDB: PAGE CONTAINS 2000. * * * VOLUME IS USER01: UNIT NAME IS F493: LOCATE RANGE.A,RANGE.B,RANGE.C,RANGE.D,RANGE.E, RANGE.F, RANGE.G,RANGE.H,RANGE.I,RANGE.J, RANGE.K,RANGE.L. END. </pre>

PRDBSのCODASYLスキーマに対するDBTGDLを以下に示す。

説 明	D D L
SCHEMA レコード型 EMPLOYEE	<pre> DEFINE LCSPROJDB; DEVICE DA; SCHEMA LISSUZUKI, M17C-F, PROJSUB4, TAKIZAWA; REC:RANGEF EMPLOYEE (END /105 / /DRA/ANA, ENAME /C10, EFNAME /C20, ESIX /C01, BIRTH-PLACE /C10, BIRTH-YEAR /102, BIRTH-MONTH /102, ALMA-MATER /C30, MAJOR /C30, DEGREE /C01, DEG-YEAR /102, HIRED-YEAR /102, RETIRED-YEAR /102, DEPT /C20, EPOSITION /C20, TEL /C12, EXT /104); </pre>
PROJECT	<pre> REC:RANGEA PROJECT (PNO /C08 /CALC/DNA/ANA, PNAME /C100, PROJ-TYPE /C01, STATUS-F /C03, SYEAR /102, EYEAR /102, BUDGET /111); </pre>
REPORTR	<pre> REC:RANGE E REPORTR (PNO /C08 /CALC/DNA/ANA, TITLE /C200, SOURCE-N /C100, VOL /C08, NUMB /103, MONTH /102, YEAR /102, FROM-PAGE /105, TO-PAGE /105, TOTAL-PAGE /105, NO-OF-AUTHOR /102, LANGUAGE /C10, WRITTEN-YEAR /102, COST /109, RTYPE /C01, RSTATE /C01); </pre>

説 明	D D L			
ORG-EMP	SET	ORG-EMP OWNER	ORGANIZATION	
SYS-ORG	SET	SYS-ORG MEMBER	EMPLOYEE /ENO	/ENO /A /OWNER/PRIOR;
SYS-SPON	SET	SYS-SPON OWNER	SYSTEM	
SYS-EMP	SET	SYS-EMP MEMBER	SPONSOR;	
		OWNER	SYSTEM	
		MEMBER	EMPLOYEE /ENO	/ENO /A;

HIPによって生成されたPRDBSのLCSを図2.7に示す。

ESR

SUBJECT (@SUBJECT, subjectn)

SPONSOR (@SPONSOR, sname, stype)

PROJECT (@PROJECT, pno, pname, proj-type,
status-f, syear, eyear, budget)

PRODUCT (@PRODUCT, pd-no, pd-name, ptype,
cost, year, language, psije)

EMPLOYEE (@EMPLOYEE, eno, ename, efname,
esex, birth-place, birth-year,
birth-momth, alma-mater, major,
degree, deg-year, hired-year,
retired-year, dept, position, tel, ext)

REPORTR (@REPORTR, rno, title, source-n, vol,
numb, month, year, from-page, to-page,
total-page, no-of-author, language,
written-year, cost, rtype, rstate)

ORGANIZATION (@ORGANIZATION, org-name, org-type)

R S R

<u>SPON-PROJ</u>	(<u>@SPONSOR</u> , <u>@PROJECT</u>)
<u>PROJ-PROD</u>	(<u>@PROJECT</u> , <u>@PRODUCT</u>)
<u>PROJ-EMP-LNK</u>	(<u>@PROJECT</u> , <u>@EMPLOYEE</u> , smonth, eyear, emonth, pposition, role, percentage)
<u>PROJ-SUBJ</u>	(<u>@PROJECT</u> , <u>@SUBJECT</u>)
<u>REP-SUBJ</u>	(<u>@REPORTR</u> , <u>@SUBJECT</u>)
<u>EMP-REP-LNK</u>	(<u>@REPORTR</u> , <u>@EMPLOYEE</u> , auth-no)
<u>PROJ-RE</u>	(<u>@PROJECT</u> , <u>PRIOR-PR-@PROJ</u>)
<u>ORG-EMP</u>	(<u>@ORGANIZATOR</u> , <u>@EMPLOYEE</u>)

☒ 2.7 PRCBS/LCS

b) DCDBS

A I M上に定義されたDCDBSのスキーマを以下に示す。

説 明	D D L
スキーマ名 DCDBS	<pre> * * * ***** * * SCHEMA ENTRY * * ***** SCHEMA NAME IS DCDBS FOR DB; AUTHOR IS M-YOKUSUKA; DATE-WRITTEN 1982-06-14; REMARKS DOCUMENTS DATABASE SYSTEMS. * ***** * * RECORD ENTRY * * ***** </pre>
レコード型名 PROJECT	<pre> RECORD NAME IS PROJECT; LOCATION MODE IS RANDOM USING PCODE DUPLICATES ARE NOT ALLOWED. 02 PCODE ; PIC 9(4). 02 PNAME ; PIC X(100). 02 PTYPE ; PIC X(2). 02 SYEAR ; PIC 9(2). 02 EYEAR ; PIC 9(2). 02 PNO ; PIC X(6). </pre>
LOC	<pre> RECORD NAME IS LOC; LOCATION MODE IS RANDOM USING DEPT DUPLICATES ARE NOT ALLOWED. 02 DEPT ; PIC X(2). 02 SECT ; PIC X(3). 02 MNAME ; PIC X(30). 02 EXT ; PIC 9(4). </pre>
DOC-AUTH	<pre> RECORD NAME IS DOC-AUTH. 02 AUTH-NO ; PIC 9(2). </pre>
AGENT	<pre> RECORD NAME IS AGENT; LOCATION MODE IS RANDOM USING ANAME DUPLICATES ARE NOT ALLOWED. 02 ANAME ; PIC X(100). 02 ADDRESS ; PIC X(100). 02 Z-CODE ; PIC X(6). 02 TEL ; PIC X(12). 02 EXT ; PIC X(4). 02 E-LNAME ; PIC X(30). 02 E-FNAME ; PIC X(30). </pre>
DOCUMENT	<pre> RECORD NAME IS DOCUMENT. 02 JNO ; PIC 9(7). 02 ISBN ; PIC X(20). 02 C-CODE ; PIC 9(4). 02 C-DESCRIPT; PIC X(10). 02 TYP ; PIC X. 02 TITLE ; PIC X(400). 02 NO-OF-AUTHOR; PIC 9(2). 02 P-YEAR ; PIC 9(2). 02 P-MONTH ; PIC 9(2). 02 P-COMPANY ; PIC X(100). 02 COST ; PIC 9(8). 02 PURCH-YEAR; PIC 9(2). 02 PURCH-MONTH; PIC 9(2). 02 PURCH-DAY; PIC 9(2). 02 FISCAL-YEAR; PIC 9(2). 02 ORDER-YEAR; PIC 9(2). 02 ORDER-MONTH; PIC 9(2). 02 ORDER-DAY; PIC 9(2). 02 ORDER-NO; PIC 9(5). 02 CODE-IN-PROJ; PIC X(10). 02 VOL ; PIC X(6). </pre>

説 明	D D L
EMPLOYEE	<pre> 02 NUMB ; PIC 9(4). 02 TPAGE; PIC 9(5). 02 LANGUAGE ; PIC X. RECORD NAME IS EMPLOYEE; LOCATION MODE IS RANDOM USING ECODE DUPLICATES ARE NOT ALLOWED. 02 ECODE ; PIC 9(4). 02 FIRST-NAME ; PIC X(30). 02 LAST-NAME ; PIC X(30). 02 DEPT ; PIC X(2). 02 EXT ; PIC 9(4). 02 END ; PIC 9(5). </pre>
DOC-EMP	<pre> RECORD NAME IS DOC-EMP. 02 LENT-DATE ; PIC 9(6). 02 RETURN-YEAR; PIC 9(2). 02 RETURN-MONTH; PIC 9(2). 02 RETURN-DAY; PIC 9(2). 02 INTEND-YEAR; PIC 9(2). 02 INTEND-MONTH; PIC 9(2). 02 INTEND-DAY; PIC 9(2). 02 STATUS ; PIC X(2). </pre>
AUTHORR	<pre> RECORD NAME IS AUTHORR. 02 FIRST-NAME ; PIC X(30). 02 MIDDLE-NAME; PIC X(10). 02 LAST-NAME ; PIC X(30). 02 NATIONALITY; PIC X(2). </pre>
セット型名 S-D	<pre> * * * * ***** # SET ENTRY * ***** SET NAME IS S-D; OWNER IS SYSTEM; MEMBER IS DOCUMENT LINKED TO OWNER PRIOR; INSERTION IS SORTED USING JND DUPLICATES ARE NOT ALLOWED. </pre>
D-A	<pre> SET NAME IS D-A; OWNER IS DOCUMENT; MEMBER IS DOC-AUTH LINKED TO OWNER PRIOR; INSERTION IS SORTED USING AUTH-NO DUPLICATES ARE NOT ALLOWED. </pre>
S-A	<pre> SET NAME IS S-A; OWNER IS SYSTEM; MEMBER IS AUTHORR LINKED TO OWNER PRIOR; INSERTION IS SORTED USING LAST-NAME DUPLICATES ARE NOT ALLOWED. </pre>
A-D	<pre> SET NAME IS A-D; OWNER IS AUTHORR; MEMBER IS DOC-AUTH LINKED TO OWNER PRIOR; INSERTION IS LAST. </pre>
E-D	<pre> SET NAME IS E-D; OWNER IS EMPLOYEE; MEMBER IS DOC-EMP LINKED TO OWNER PRIOR; INSERTION IS SORTED USING LENT-DATE DUPLICATES ARE LAST. </pre>
D-E	<pre> SET NAME IS D-E; OWNER IS DOCUMENT; MEMBER IS DOC-EMP LINKED TO OWNER PRIOR; INSERTION IS SORTED USING LENT-DATE DUPLICATES ARE NOT ALLOWED. </pre>

説明	D	D	L
AGED	* SET NAME IS AGE-D; OWNER IS AGENT; MEMBER IS DOCUMENT LINKED TO OWNER PRIOR; INSERTION IS SORTED USING JMI DUPLICATES ARE NOT ALLOWED.		
LOC-D	* SET NAME IS LOC-D; OWNER IS LOC; MEMBER IS DOCUMENT LINKED TO OWNER PRIOR; INSERTION IS SORTED USING JMD DUPLICATES ARE NOT ALLOWED.		
P-D	* SET NAME IS P-D; OWNER IS PROJECT; MEMBER IS DOCUMENT LINKED TO OWNER PRIOR; INSERTION IS SORTED USING JMI DUPLICATES ARE NOT ALLOWED.		
レンジ名 RANGEA	* * * * * * RANGE ENTRY * * * RANGE NAME IS RANGEA; RECORD NAME IS DOCUMENT; PRIME IS 150; LOGICAL PAGE IS 6; EXCLUSIVE UNIT IS RANGE.		
RANGEB	* RANGE NAME IS RANGEB; RECORD NAME IS PROJECT; PRIME IS 2; LOGICAL PAGE IS 6; EXCLUSIVE UNIT IS RANGE.		
RANGEC	* RANGE NAME IS RANGEC; RECORD NAME IS LOC; PRIME IS 1.		
RANGED	* RANGE NAME IS RANGED; RECORD NAME IS AGENT; PRIME IS 2; LOGICAL PAGE IS 6.		
RANGEE	* RANGE NAME IS RANGEE; RECORD NAME IS EMPLOYEE; PRIME IS 5; OVERFLOW IS 1; LOGICAL PAGE IS 2.		
RANGEF	* RANGE NAME IS RANGEF; RECORD NAME IS AUTHORITY; PRIME IS 80; OVERFLOW IS 20; LOGICAL PAGE IS 2.		
RANGEG	* RANGE NAME IS RANGEG; RECORD NAME IS DUC-EMP; PRIME IS 70.		
RANGEH	* RANGE NAME IS RANGEH; RECORD NAME IS DUC-AUTH; PRIME IS 65.		
データセット名 DCDBS	* * * * DATA SET ENTRY * * * DATASET NAME IS DCDBS; PAGE CONTAINS 2100.		

説 明	D D L
	<pre> * * * VOLUME IS WORK01; UNIT NAME IS F493; LOCATE RANGEA,RANGEB,RANGEC,RANGED,RANGE, RANGEF,RANGEG,RANGEH. * * * END. </pre>

DCDBSのCODASYLスキーマに対するDBTGDLを以下に示す。

説 明	D D L
SCHEMA レコード型	<pre> DEFINE LCSDCDBS: DEVICE DA: SCHEMA LISSUZUKI, M170-F, DCSUB, TANIZAWA; REC:RANGEE EMPLOYEE </pre>
EMPLOYEE	<pre> (CUCODE /104 /CALC/DNA/NNA, FIRST-NAME /C30, LAST-NAME /C30, DEPT /C02, EXT /104, END /105); </pre>
PROJECT	<pre> REC:RANGEB PROJECT (PCODE /104 /CALC/DNA/NNA, PNAME /C100, PTYPE /C02, SYEAR /102, EYEAR /102, PNO /C08); </pre>
LOC	<pre> REC:RANGEC LOC (DEPT /C02 /CALC/DNA/NNA, SECT /C03, MNAME /C30, EXT /104); </pre>
AUTHORR	<pre> REC:RANGEF AUTHORR (FIRST-NAME /C30 / / /NNA, MIDDLE-NAME /C10, LAST-NAME /C30, NATIONALITY /C02); </pre>
AGENT	<pre> REC:RANGED AGENT (ANAME /C100 /CALC/DNA/NNA, ADDRES /C100, Z-CODE /C06, TEL /C12, EXT /C04, E-LNAME /C30, E-FNAME/C30); </pre>
DOC-AUTH	<pre> REC:RANGEH:LINK DOC-AUTH (AUTH-NUM /102 / / /NNA); </pre>
DOC-EMP	<pre> REC:RANGEG:LINK DOC-EMP (CLIENT-DATE /106/ / /NNA, RETURN-YEAR/102, RETURN-MONTH/102, RETURN-DAY/102, INTEND-YEAR/102, INTEND-MONTH/102, INTEND-DAY/102, STATUS /C02); </pre>

説 明	D D L			
DOCUMENT	REC:RANGEA	DOCUMENT CJND /107 / / / /NNK, ISBN /C20, C-CODE /104, C-DESCRIPT /C10, TYP /C01, TITLE /C400, NO-OF-AUTHOR /102, P-YEAR /102, P-MONTH/102, P-COMPANY /C100, COST /108, PURCH-YEAR /102, PURCH-MONTH /102, PURCH-DAY /102, FISCAL-YEAR /102, ORDER-YEAR /102, ORDER-MONTH /102, ORDER-DAY /102, ORDER-NO /105, CODE-IN-PROJ /C10, VOL /C06, NUMB /104, TPAGE /105, LANGUAGE /C01);		
セット型 S-D	SET	S-D	OWNER MEMBER	SYSTEM DOCUMENT /JND /JND /A /OWNER,PRIOR;
D-A	SET	D-A	OWNER MEMBER	DOCUMENT DUC-AUTH /AUTH-NO /AUTH-NO/A/OWNER,PRIOR;
S-A	SET	S-A	OWNER MEMBER	SYSTEM AUTHRR /LAST-NAME/LAST-NAME/A/OWNER,PRIOR;
A-D	SET	A-D	OWNER MEMBER	AUTHRR DUC-AUTH / / / /OWNER,PRIOR;
E-D	SET	E-D	OWNER MEMBER	EMPLOYEE DOC-EMP /LENT-DATE/LENT-DATE/A/OWNER,PRIOR;
D-E	SET	D-E	OWNER MEMBER	DOC-EMP /LENT-DATE/LENT-DATE/A/OWNER,PRIOR;
AGE-D	SET	AGE-D	OWNER MEMBER	AGENT DOCUMENT /JND /JND /A /OWNER,PRIOR;
LOC-D	SET	LOC-D	OWNER MEMBER	LOC DOCUMENT /JND /JND /A /OWNER,PRIOR;
P-D	SET	P-D	OWNER MEMBER	PROJECT DOCUMENT /JND /JND /A /OWNER,PRIOR;

HIPによって生成されたDCDBSのLCSを図2.8に示す。

ESR

<u>EMPLOYEE</u>	(@EMPLOYEE, ecode, first-name, last-name, dept, ext, eno)	)
<u>PROJECT</u>	(@PROJECT, pcode, pname, ptype, syear, eyear, pno)	)
<u>LOC</u>	(@LOC, dept, sect, mname, ext)	)
<u>AUTHORR</u>	(@AUTHORR, first-name, middle-name, last-name, nationality)	)
<u>AGENT</u>	(@AGENT, aname, address, z-code, tel, ext, e-lname, e-fname)	)
<u>DOCUMENT</u>	(@DOCUMENT, jno, isbn, c-code, c-descript, typ, titie, no-of-author, p-year, p-month, p-company, cost, purch-year, purch-month, purch-day, fiscal-year, order-year, order-month, order-day, order-no, code-in-proj, vol, numb, tpage, language)	)

RSR

<u>DOC-AUTH</u>	(@DOCUMENT, @AUTHORR, auth-no)	)
<u>AGE-D</u>	(@AGENT, @DOCUMENT)	)
<u>LOC-D</u>	(@LOC, @DOCUMENT)	)
<u>P-D</u>	(@PROJECT, @DOCUMENT)	)

DOC-EMP

(@DOCUMENT, @EMPLOYEE, lent-date,
return-year, return-month, return-
day, intend-year, intend-month,
intend-day, status)

図 2.8 DCDBSのLCS

2.3 QTP (問合せ変換プロセッサ) の実現

問合せ変換プロセッサは、RQL (Relational query language) 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RWSに格納する。問合せ変換プロセッサの概要は図 2.9 の様である。

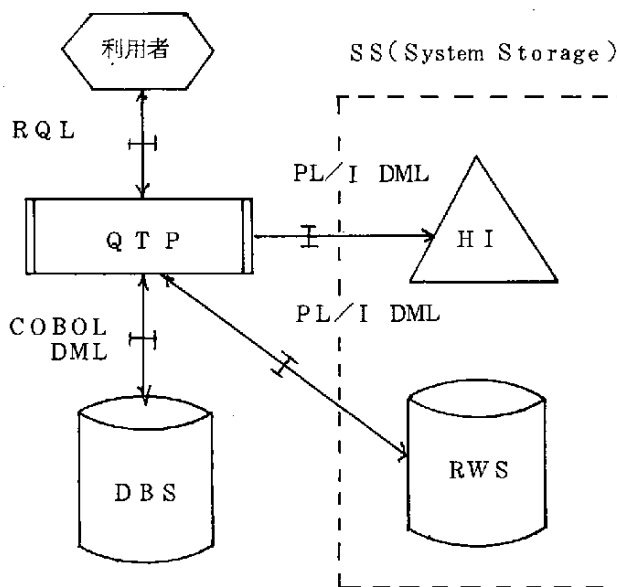


図 2.9 QTP の処理概要

LDP-V3では、集積関数を含む問合せからCOBOL DMLプログラムを生成する。集積関数を含んだ問合せに対するQTPの処理の流れは図 2.10 の様である。

G-treeは、集積関数を含む問合せを処理する為に新しく定義されるtreeである。まず集積関数の処理方法とG-treeの定義について述べ、次に集積関数を含む問合せの変換処理の実現について述べる。

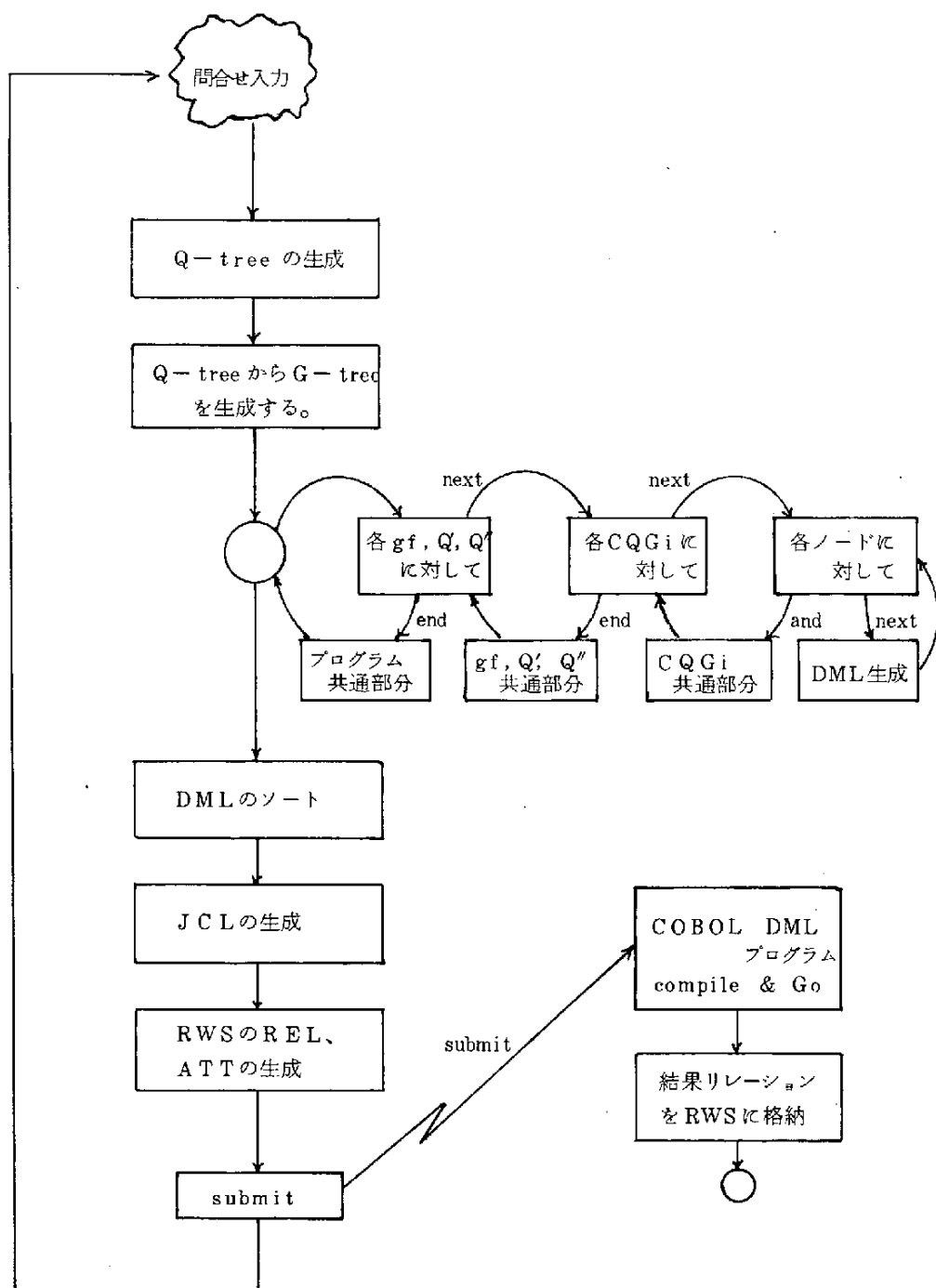
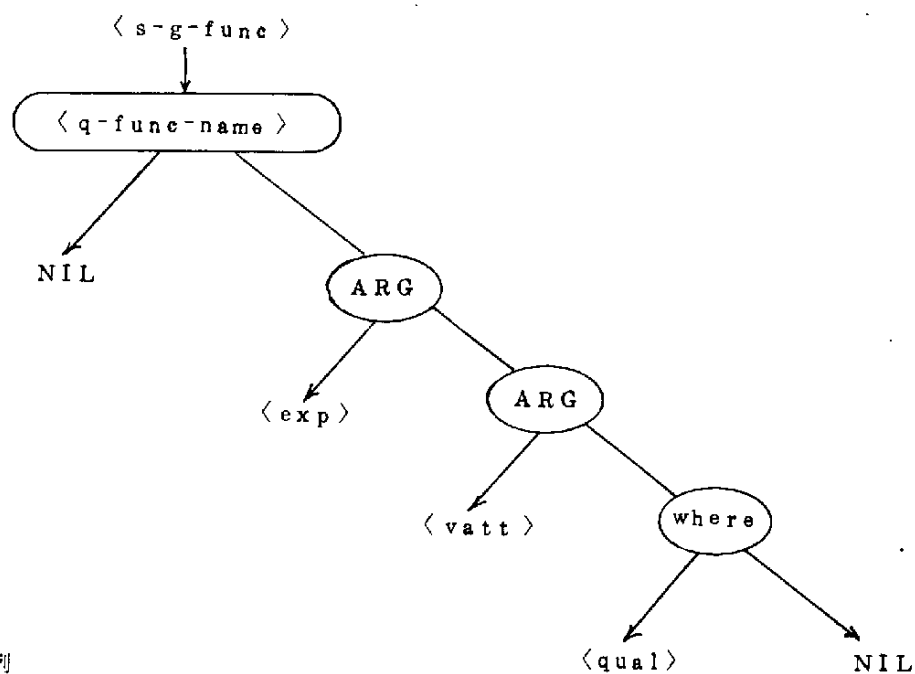


図2.10 QTPの処理の流れ

2.3.1 単純集積関数

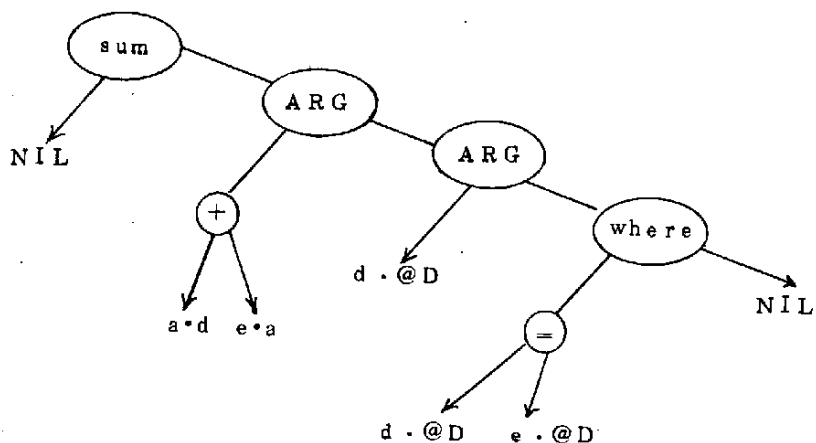
A. 構文 $\langle s\text{-}g\text{-}func \rangle ::= \langle g\text{-}func\text{-}name \rangle$
 $\langle (exp) [, \langle vatt \rangle$
 $\{, \langle vatt \rangle \} \}$
 $[\text{where } \langle qual \rangle])$

この木構造を、下図に示す。



例

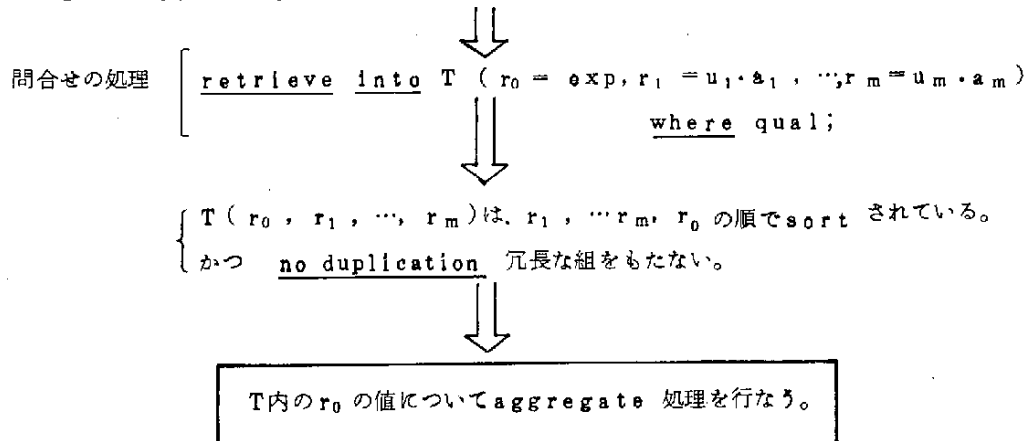
sum(d . a + e . a , d . @D where d . @D = e . @D)



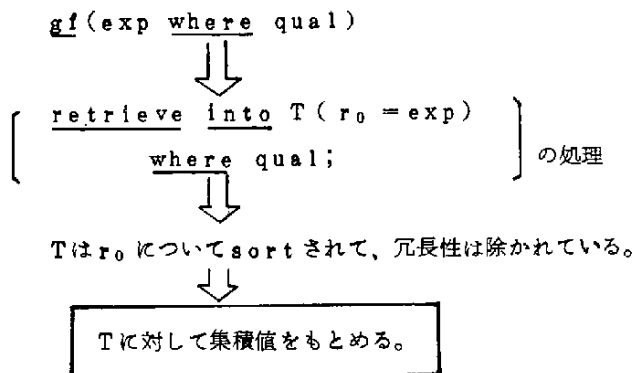
B. 単純集積関数の意味

単純集積関数は、関数内でローカルに処理を行なうことができ、結果として、スカラー値を出
力する。

(1) $gf(exp, u_1 \cdot a_1, \dots, u_m \cdot a_m \text{ where } qual)$



(2) ユニーク変数のない時



2.3.2 group-by 集積関数 (GF)

次に group-by 集積関数 ($\langle g-func \rangle$) を考える。group-by 集積関数は、単純集
積関数と異なり関数単独では処理できない。

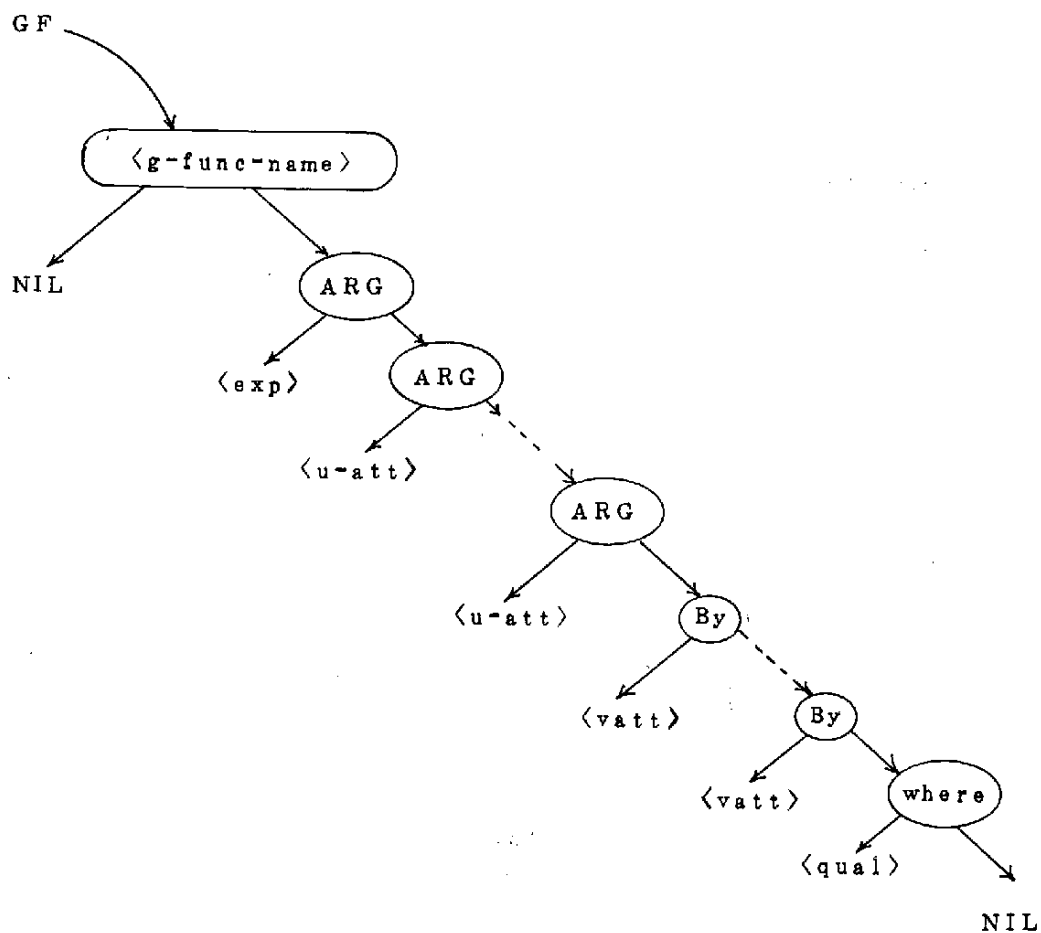
A. 構文

```

<g-func> ::= <g-func-name> ( <exp> [, <u-att> {, <u-att> } ] :
                               by <vatt> { by <vatt> } [ where qual ] )
  
```

$\{ \langle vatt \rangle : \text{group-by 変数}$
 $\{ \langle u-att \rangle : \text{ユニーク 変数}$

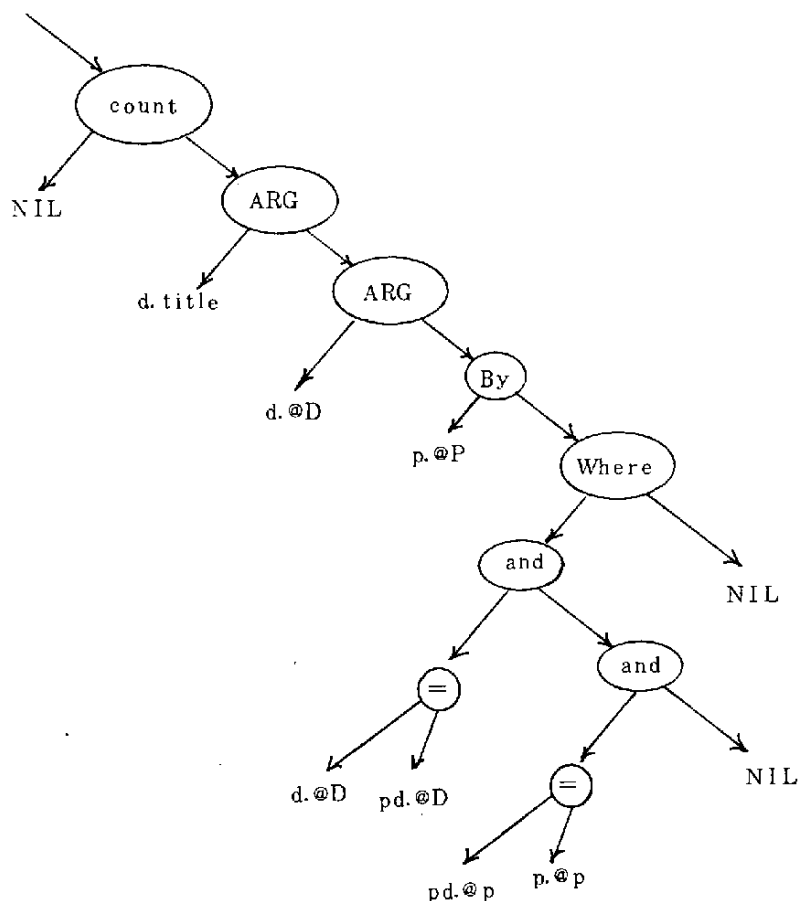
〈g-func〉(GF)の木構造は以下の様である。



例 count (d. title, d.@D by p.@P

where d.@D=pd.@D and

pd.@P=p.@P)



B. group-by 集積の意味

(1) gf(exp, $x_1 \cdot a_1, \dots, x_m \cdot a_m$ by $y_1 \cdot b_1 \dots$ by $y_n \cdot b_n$ where qual)



$T(t_1, \dots, t_n, u_1, \dots, u_m, t)$ を求める。

retrieve into $T(t_1 = y_1 \cdot b_1, \dots, t_n = y_n \cdot b_n,$
 $u_1 = x_1 \cdot a_1, \dots, u_m = x_m \cdot a_m,$
 $t = \text{exp})$

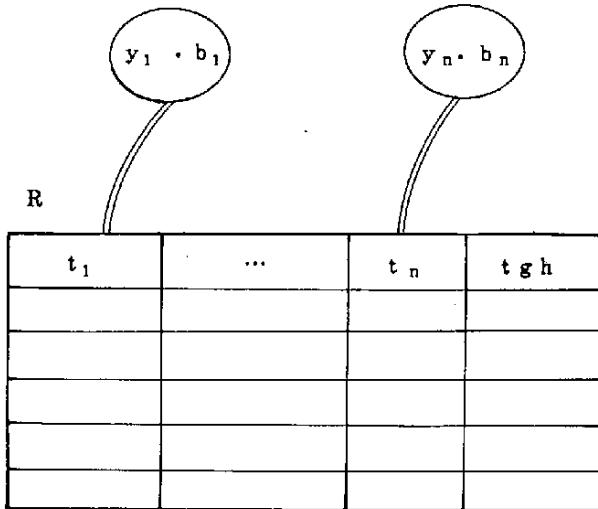
where qual ;

{ T は、 $t_1, \dots, t_n, u_1, \dots, u_m, t$ の順で sort されている。又、冗長な組をもたない。



属性 $t_1, \dots, t_n$ の各値の組に対して、 t の値集合に対して集積演算を行なう。この値を tgf とする。

$R(t_1, \dots, t_n, tgf)$ なる関係が結果となる。 $R \cdot t_1, \dots, R \cdot t_n$ は、問合せ内の aggregate 関数の外部から規定される。



$x_1 \cdot a_1, \dots, x_m \cdot a_m$ は uniqueness の為の属性である。

$y_1 \cdot b_1, \dots, y_n \cdot b_n$ は group-by の為の属性である。

これらは、aggregate 関数 gf の外部から規定される。

問合せ (Q) 内の式 exp が集積関数 $gf_1, \dots, gf_k$ を参照するとき

```

exp ( gf1 ( exp1, ua11, ..., ua1m by va11 ... by va1n1,
          where qual ),
      gfk ( expk, uak1, ..., uakmk by vak1 ... by vaknk
          where qual ) )

```



ここで $v_{a_{ij}} = v_{ij} \cdot a_{ij}$ とする。

○ 各 group-by 集積関数 gf_i を gf_i の結果関係 R_i の組変数 r_i に置き換える。

即ち、 $\exp(R_k, r_k, \dots, R_1, r_1)$

○Qのqualに各 gf_i 毎にgroup-by 変数 va_{ij} の結合式を接合する。

$$qual' \leftarrow qual \bigwedge_{i=1}^k (R_i \cdot t_{i1} = v_{i1} \cdot a_{i1} \wedge \dots \wedge R_i \cdot t_{in_i} = v_{in_i} \cdot a_{in_i})$$

接合された結合式をgroup-by結合式と呼ぶ。

C. 集積演算結果のタイプ、長さ

集積演算 (count, any, max, min, sum, aver) の結果値のタイプ長さを次の表にまとめる。

集 積 演 算	ダ イ ブ	長 さ
count	I	4 バイト
any	I	4
max	exp の型	4
min	"	4
sum	"	4
aver	"	4

$$\begin{cases} = 1 : \text{exist} \\ = 0 : \text{non exist} \end{cases}$$

I ≡ 整数

D. DML化

(1) 集積関数 gf_i 毎に結果関係 $R_i(r_{i1}, \dots, r_{ik_i}, r)$ を生成する。

r = 集積関数の値

$r_{i1}, \dots, r_{ik_i}$ = group-by 属性

(2) 問合せの本体からも結果関数 $R(\dots, a_{i1}, \dots, a_{ik_i}, \dots)$ を生成する。

R は複数の $DQG_j (j=1, \dots, \ell)$ から成っている時はこれらの個々を考える。

(3) DQG_j 間の結合処理に $R_i (i=1, \dots, r_n)$ との結合処理も同時に行わせる。

例 Q retrieve into $R(r_1 = \text{aver}(x \cdot a + 5, x \cdot b / y \cdot e, x \cdot f$
where $x \cdot a = y \cdot k),$
 $r_2 = z \cdot f)$

where

$x \cdot h = y \cdot d \wedge$

$z \cdot f \geq \text{count}(x \cdot h / x \cdot a);$

$gf_1 \quad R_1 (r_{11} (y \cdot e), r_{12} (x \cdot f), r_1)$
 $r_1 = \text{aver} (x \cdot a + 5, x \cdot b \text{ by } y \cdot e \text{ by } x \cdot f$
 $\quad \text{where } x \cdot a = y \cdot h)$
 $gf_2 \quad R_2 (r_{21} (x \cdot a), r_2)$
 $r_2 = \text{count} (x \cdot h \text{ by } x \cdot a);$
 $Q' \quad \text{retrieve into } R (r_1 = R_1 \cdot r_1, r_2 = z \cdot f)$
 where
 $x \cdot h = y \cdot d \wedge z \cdot f \geq R_2 \cdot r_2 \wedge$
 $R_1 \cdot r_{11} = y \cdot c \wedge R_1 \cdot r_{12} = x \cdot f \wedge$
 $R_2 \cdot r_{21} = x \cdot a;$

DML

	$R_1 (r_{11}, r_{12}, r_1)$ の生成
	$R_2 (r_{21}, r_2)$ の生成
	$CQG_1 \quad T_1 (x \cdot a, x \cdot f, y \cdot e)$ の生成
	$CQG_2 \quad T_2 (z \cdot f)$ の生成
	R_1, R_2, T_1, T_2 の結合処理

$$\begin{aligned}
 R (r_1 = R_1 \cdot r_1, r_2 = T_2 \cdot f) \leftarrow \begin{pmatrix} R_1 \cdot r_{11} = T_1 \cdot e \wedge \\ R_1 \cdot r_{12} = T_1 \cdot f \wedge \\ R_2 \cdot r_{21} = T_1 \cdot a \wedge \\ R_2 \cdot r_{12} \leq T_2 \cdot f \end{pmatrix}
 \end{aligned}$$

E. 集積関数の例

(1) 自分の部の平均給料よりも多くて、会社全体の平均よりも多いものを求めよ。

$\text{range} (e, \text{EMPLOYEE});$
 $\text{retrieve into } R (\text{name} = e \cdot \text{ename})$
 where
 $e \cdot \text{sal} \geq \text{aver} (e \cdot \text{sal}, e \cdot @E \text{ by } e \cdot \text{dept}) \text{ and}$
 $e \cdot \text{sal} \geq \text{aver} (e \cdot \text{sal}, e \cdot @E);$

単純集積

- (2) 新従業員の eno を、現在の最大値より 1 つ多くして追加せよ。

```

range (e, EMPLOYEE);
append to EMPLOYEE
      単純集積
      (eno = max (e.eno) + 1,
       ename = ..... );

```

- (3) project 毎に、メンバの平均年齢を求めよ

```

range (e, EMPLOYEE) (pe, PROJ-EMP-LNK)
  (p, PROJECT);
retrieve into R (p.pno, average-age =
  aver(82 - e.birth-year by p.pno
  where
    e.@E=pe.@E and pe.@P=p.@P));

```

- (4) (3)で、project が 78 以降のもの

```

retrieve into R (p.pno
  average-age = aver(82 - e.birth-year by p.pno
  where e.@E=pe.@E and
    pe.@P=p.@P and
    p.syear ≥ 82 ));

or

retrieve into R (p.pno)
  average-age = aver (82 - e.birty-year by p.pno
  where e.@E=pe.@E and pe.@P=p.@P))
where
  p.syear ≥ 82;

retrieve into R (p.pno,
  average-age = aver (82 - e.birth-year by p.pno
  whear e.@E=pe.@E and
    pe.@p=p.@p and
    p.syear ≥ 82 ));

```

2.3.3 G-tree

A. 定義

G-tree は、Q-tree から生成される。問合せ (Q) が ℓ 個の集積関数 $ag_1, \dots, ag_\ell$ を持つとする。

$$\Omega_{ga} \triangleq \{ ag_1, \dots, ag_\ell \} = \text{集積関数の集合}$$

$\Omega_{sa} \triangleq$ 単純集積関数の集合 $\triangleq \{sa_1, \dots, sa_n\}$

$\Omega_{gf} \triangleq$ group-by 集積関数の集合 $\triangleq \{gf_1, \dots, gf_m\}$

$$\left(\begin{array}{l} \text{即ち, } \Omega_{ga} = \Omega_{sa} \cup \Omega_{gf} \\ \Omega_{sa} \cap \Omega_{gf} = \emptyset \end{array} \right)$$

単純集積関数

$sa_i \triangleq sa_i (exp, x_{i1} \cdot a_{i1}, \dots, x_{ik_i} \cdot a_{ik_i} \text{ where qual})$

group-by 集積関数

$gf_j \triangleq gf_j (exp, x_{j1} \cdot a_{j1}, \dots, x_{jk_j} \cdot a_{jk_j} \text{ by } y_{j1} \cdot b_{j1}, \dots, y_{jh_j} \cdot b_{jh_j} \text{ where qual})$

$Agf_i \triangleq \{y_{i1} \cdot b_{i1}, \dots, y_{ih_i} \cdot b_{ih_i}\} \triangleq$ group-by 変数の集合

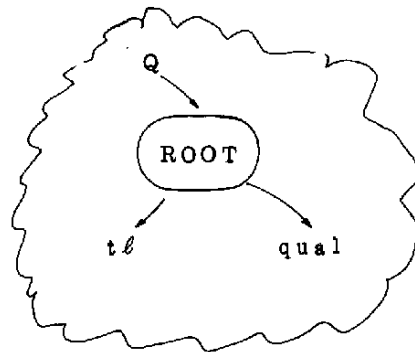
ここで以下の置換 σ を定義する。

置換 $\sigma = \{pv_1 / sa_1, \dots, pv_n / sa_n\}$

ここで pv_i は、ある変数である。

以下の問合せ Q を考える。

$Q: \text{retrieve into } Q (r_1 = exp_1, \dots, r_k = exp_k) \\ \text{where qual ;}$



この時、以下を定義する。

$tl \triangleq \{r_1 = exp_1, \dots, r_k = exp_k\} \triangleq Q$ の目標リスト

$\Omega_{Rgf} \triangleq \{r_i = exp_i \mid r_i = exp_i \in \Omega_R \wedge \\ exp_i \text{ が } gf \text{ を参照している}\}$

$= tl$ の中で、 exp_i が group-by 集積関数を参照しているものの集合

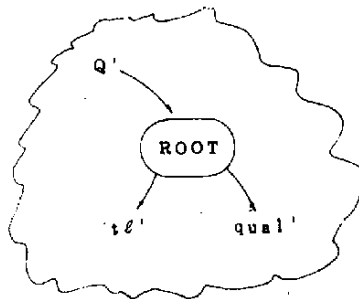
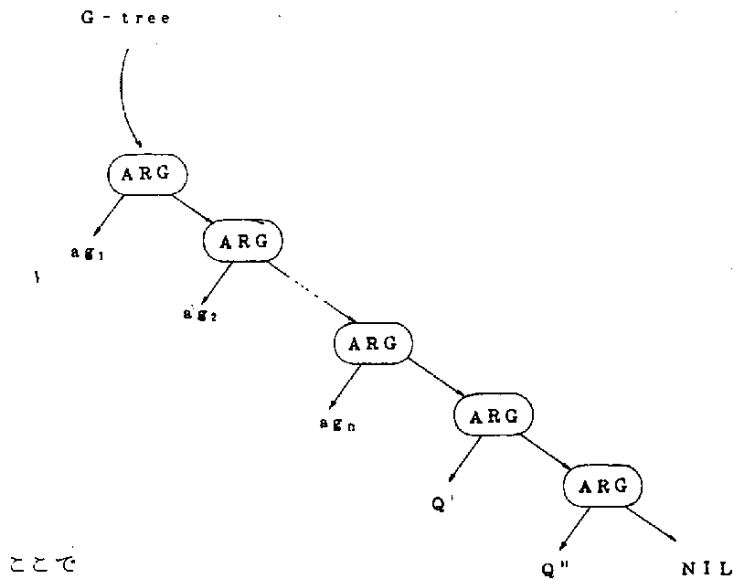
$Argf \triangleq \{x \cdot a \mid x \cdot a = exp_i \in \Omega_{Rgf} \text{ が参照している variable}\}$

$qual \triangleq C_1 \wedge \dots \wedge C_q \triangleq Q$ の条件式

$\Omega_{\pi gf} \triangleq \{C_i \in qual \mid C_i \text{ は } gf \text{ を参照している}\}$

$\triangleq Q$ 内で、group-by 集積関数を参照している C_i の集合

Qに対するG-treeは、次の様になる。



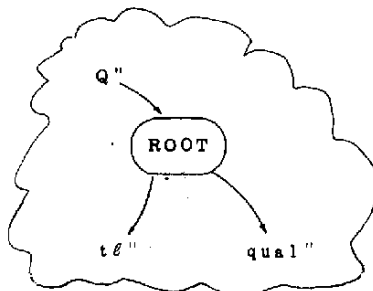
$$tel' \triangleq \sigma(tel - \Omega_{Rgf})$$

$\triangleq$ 目標リスト内の要素($r_i = exp_i$)の内で exp_i が gf を含まないもののうち単純集積関数(sa_j)を記号(pv_j)で置換える。

$$qual' \triangleq \alpha(qual - \Omega_{ggf})$$

$\triangleq$ $qual$ 内の各conjunct(C_i)の内で gf を参照していないもののうち、単純集積関数(sa_j)を参照しているものを、 pv_j に置き換えたものの積。

Q'' は、各group-by集積関数の結果のgroup-by変数による結合を表している。



$$\begin{aligned}
t\ell'' &\triangleq \{ r_1 = \text{exp}_1, \dots, r_h = \text{exp}_h \} \\
&= \{ \sigma'_1 / \text{exp}_1, \dots, \sigma'_k / \text{exp}_k \} \\
\sigma''_i &= \begin{cases} Q'.r_i & \text{if } r_i = \text{exp}_i \in \Omega_{\text{ggf}} \\ \sigma''_{\text{exp}_i} & \text{otherwise} \end{cases} \\
\sigma'' &= \{ R_1.r_1 / \text{gf}_1, \dots, R_m.r_m / \text{gf}_m \} \\
\text{qual}'' &\triangleq \sigma'' \Omega_{\text{ggf}} (\wedge \sigma'' C) \\
&\quad \forall C \in \Omega_{\text{ggf}} \\
\wedge \quad &\text{rel}(\text{gf}_i).r_{ij} = Q'.\text{rat}(y_j.a_j) \\
\forall \text{gf} \in \Omega_{\text{gf}} \\
\text{rel}(\text{gf}_i) &\triangleq \text{gf}_i \text{ の結果関係名} \\
\text{rat}(y_i.a_i) &\triangleq y_i.a_i \text{ に対して } Q'' \text{ で生成された結果属性名}
\end{aligned}$$

2.3.4 QTPの実現

QTPは、集積関数を含まない問合せを処理する問合せ変換プロセッサを拡張して実現された。Q-tree からG-tree を生成するGTGとG-tree から集積関数又はQ'に対するQ-tree を取り出してDMLを生成するGDMLGが追加された。GDMLGでは、1つのQ-tree を取り出し、構造変換、アクセスパス生成、DML生成を行なう処理モジュールが呼び出されて処理を実行する。QTPの処理概要は図2.11の様である。

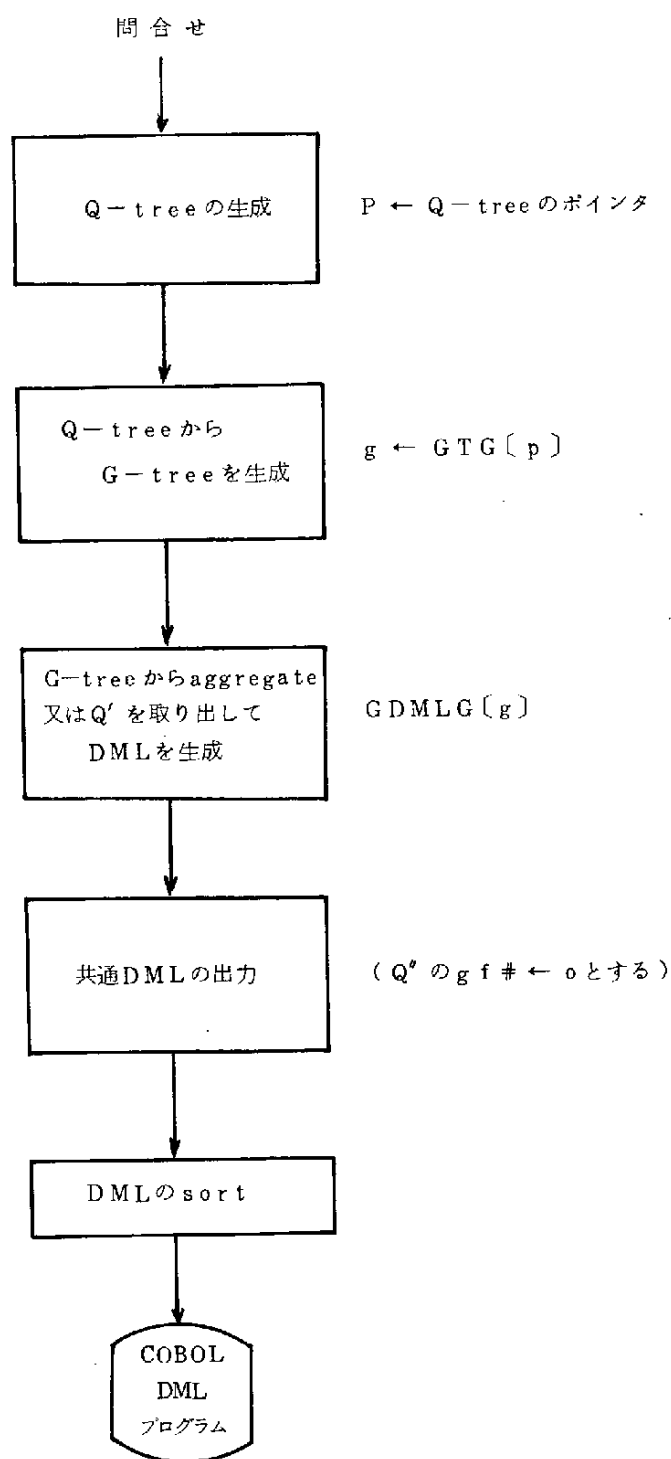


図 2.11 QTPの処理概要

A. Q-tree の生成

Q-tree の生成では、利用者が入力した RQL 問合せの文字列から、内部表現として 2 分木 (Q-tree) と 3 つのテーブル (RRTBL、RTBL、RATBL) を生成する。Q-tree 生成の処理概要は図 2.12 の様である。

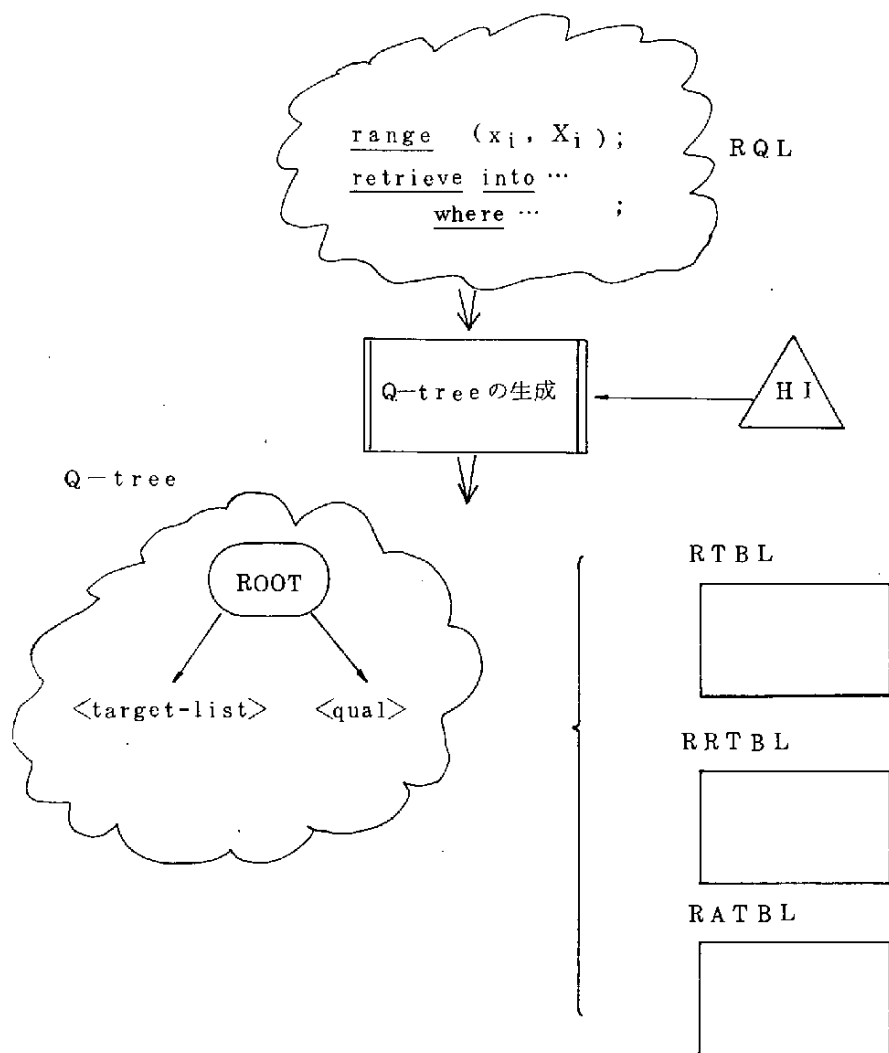
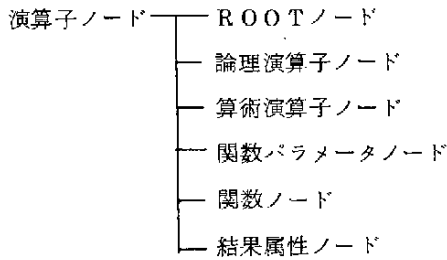
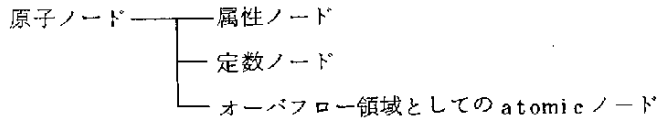


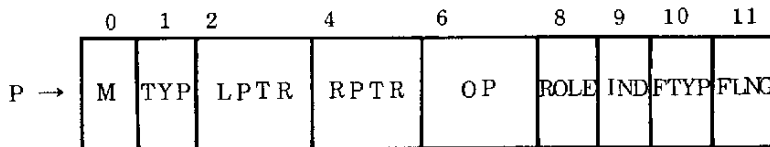
図 2.12 Q-tree 生成の処理概要

Q-tree は、主記憶内の連続領域 (自由ノード空間 FNS) の連続した 3 ワード (12 バイト) のノードで構成される。

ノードの種類は大別すると、原子ノードと演算子ノードに分けられる。演算子ノードは主に、2 つのポインタ・フィールドと演算子フィールドから成る。これに対して原子ノードは、1 つのポインタ・フィールドと原子情報フィールドから成る。



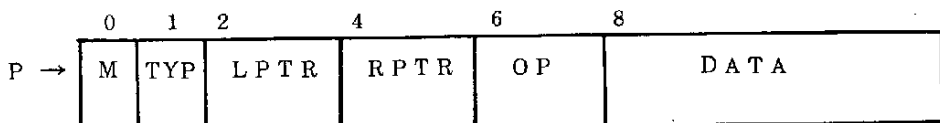
i) 属性ノードの構造



P: ポインタ

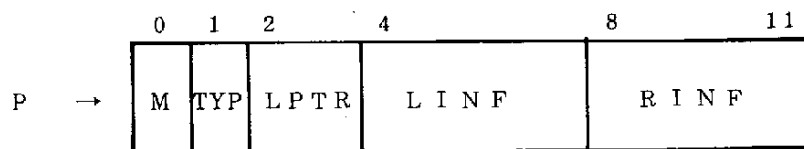
フィールド名	説 明
T Y P	TYP (P) = 0 (属性ノードであることを示す。)
L P T R	tuple - substitute時に、この属性がとる値を格納した定数ノードへのポインタが入る。それ以外ではNILである。
R P T R	RTBL (range table) へのポインタが入る。
O P	この属性の属性番号 (ATTN O) が入る。
R O L E	この属性のリレーション内での役割を示す。 $\text{ROLE (P)} = \begin{cases} 0 \text{ (NKEY) } \cdots \cdots \text{ non key} \\ 10 \text{ (K E Y) } \cdots \cdots \text{ primary key} \\ 20 \text{ (CKEY) } \cdots \cdots \text{ composite key} \end{cases}$
I N D	この属性上のインデックス情報
F T Y P	この属性のとり得る値のデータ・タイプを示す。 (整数タイプ = 2、文字タイプ = 1)
F L N G	この属性のとり得る値の長さを示す。(0 ~ 63)

ii) 定数ノードの構造



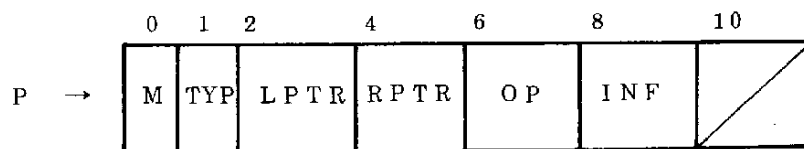
フィールド名	説 明
T Y P	TYP(P) = 1 (定数ノードであることを示す)
LPTR	多倍長データを格納する場合、次のCNTNへのポインタ。 他の場合はNIL
RPTR	DATA部に入る値の有効長
OP	値のタイプが入る。 $OP(P) = \begin{cases} 1(C) \cdots \cdots \text{文字} \\ 2(D) \cdots \cdots \text{整数} \end{cases}$
DATA	値が格納される。 整数のときは $10^{10} - 1$ までの値、文字のときは4文字まで格納できる。

iii) 原子ノードの構造



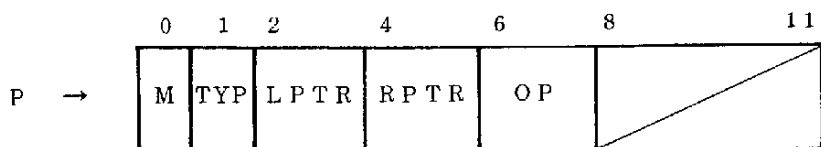
フィールド名	説 明
T Y P	TYP(P) = 2 (原子ノードであることを示す)
LPTR	次のATMNへのポインタ
LINF	任意の情報が格納される。
RINF	詳細は、AND/ROOTノードの項に記述する。

iv) 結果属性ノードの構造



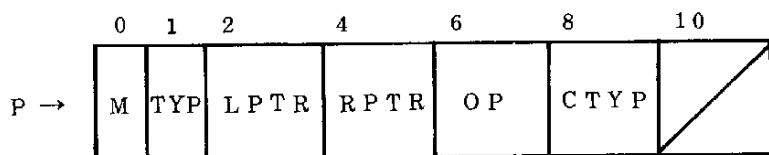
V) 単項演算子ノードの構造

NOTノード、算術関数ノード、集積関数ノード



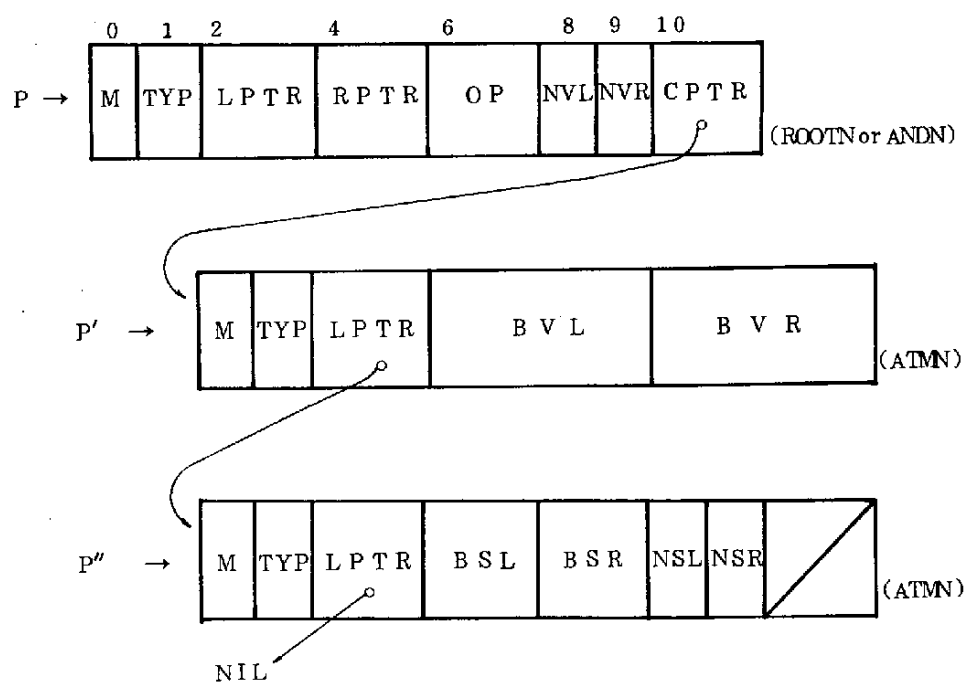
フィールド名	説 明	
T Y P	TYP(P) = 5 (=OPN)	
LPTR	LPTR(P) = NIL	
RPTR	ポインタ。関数の場合は、パラメータ・リストへのポインタ	
OP	OP(P) = NOT (2 0 0) ANY (1 0 0 0) AVER (1 0 0 1) COUNT (1 0 0 2) SUM (1 0 0 3) MAX (1 0 0 4) MIN (1 0 0 5) } group-by 集積関数 ANY (1 0 2 0) AVER (1 0 2 1) COUNT (1 0 2 2) SUM (1 0 2 3) MAX (1 0 2 4) MIN (1 0 2 5) } 単純集積関数 ABS (1 1 0 0) MOD (1 1 0 1) } 算術関数	

VI) 比較演算子ノードの構造



フィールド名	説 明
T Y P	TYP(P)=5 (=OPN)
LPTR	1st operandへのポインタ
RPTR	2nd operandへのポインタ
OP	OP(P)=LT(<) (300) LE(≤) (301) EQ(=) (302) GE(≥) (303) GT(>) (304) NE(≠) (305)
CTYP	CTYP(P)=KKJ (0) } KNJ (1) } join clause NKJ (2) } NNJ (3) } KR (10) } NR (11) } restriction clause

VD ROOTノード、ANDノードの構造



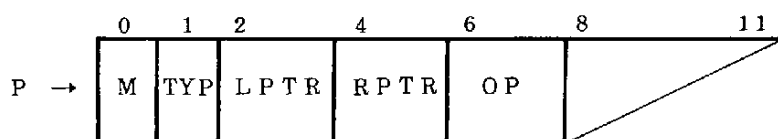
フィールド名	説 明
T Y P	TYP(P) = 5 (=OPN)
LPTR	ANDN : 1st operand へのポインタ ROOTN: target-list tree へのポインタ
RPTR	ANDN : 2nd operand へのポインタ ROOTN: qualification tree へのポインタ
OP	OP(P) = AND (100) ROOT (30)
N V L	このノードの左手の subtree 内の変数の数 (≤ 32)
N V R	このノードの右手の subtree 内の変数の数 (≤ 32)
CPTR	CPTR(P) = P'
B V L	BVL(P') = 左手の subtree 内の変数の bit-map (32ビット)
B V R	BVR(P') = 右手の subtree 内の変数の bit-map (32ビット)
B S L	BSL(P'') = 左手の subtree 内のサイトの bit-map (16ビット)
B S R	BSR(P'') = 右手の subtree 内のサイトの bit-map (16ビット)
N S L	NSL(P'') = 左手の subtree 内のサイト数 (≤ 16)
N S R	NSR(P'') = 右手の subtree 内のサイト数 (≤ 16)

Ⅶ) 算術演算子ノード

論理演算子 (OR) ノード

関数パラメータ (BY, ARG, WHERE) ノード

の構造



フィールド名	説 明
T Y P	TYP(P) = 5 (=OPN)
LPTR	1st operand へのポインタ
RPTR	2nd operand へのポインタ
OP	OP(P) = OR (101) PLUS (400) TIMS (600) DIV (601) POW (700) BY (1300) WHERE (1400) ARG (1200)

表 2.10 R R T B L

RRTBLは、QTPコマンド(define, retrieve append, delete, replace)の結果関係(視野も含める)の情報を保持する。

属 性 名	属性番号	ROLE	TYPE	LENGTH	説 明
rrel-name	1	K	C	16	結果関係名又は視野名
degree	2	N	D	2	結果関係の属性
type	3	N	C	2	V = define D = delete R = retrieve RP = replace A = append
UP-REL	4	N	C	16	type がVの場合、視野スキーマ名。 更新の場合、更新される関係名
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)
apt	7	N	P	2	この問合せのQ-treeへの pointer
aptV	8	N	P	2	query modification された Q-treeへのpointer

表 2.11 R T B L

R T B L (range table) は、組変数と関係との対応表である。

属 性 名	属性番号	ROLF	TYPE	LENGTH	説 明
var	1	K	C	2	組変数名
rel-name	2	N	C	16	var に対応する関係名
type	3	N	C	2	V : 視野 L : les 関係 R : snap-shot
work	4	N	D	1	work area
type	5	N	D	1	1 : ESR 2 : RSR 9 : snap-shot 又は 視野
qptv	6	N	P	2	視野 又は snap-shot の Q- tree へのポインタ

表 2.12 R A T B L

R A T B L は、結果関係 (視野も含む) 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	結果関係名又は視野名
att-no	2	K	D	2	属性番号
att-name	3	N	C	16	属性番号
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	定義域名

B. GTG (G-tree 生成) の実現

GTGは、問合せに対するQ-treeを入力してG-treeを生成する。GTGの処理概要は図2.13の様である。

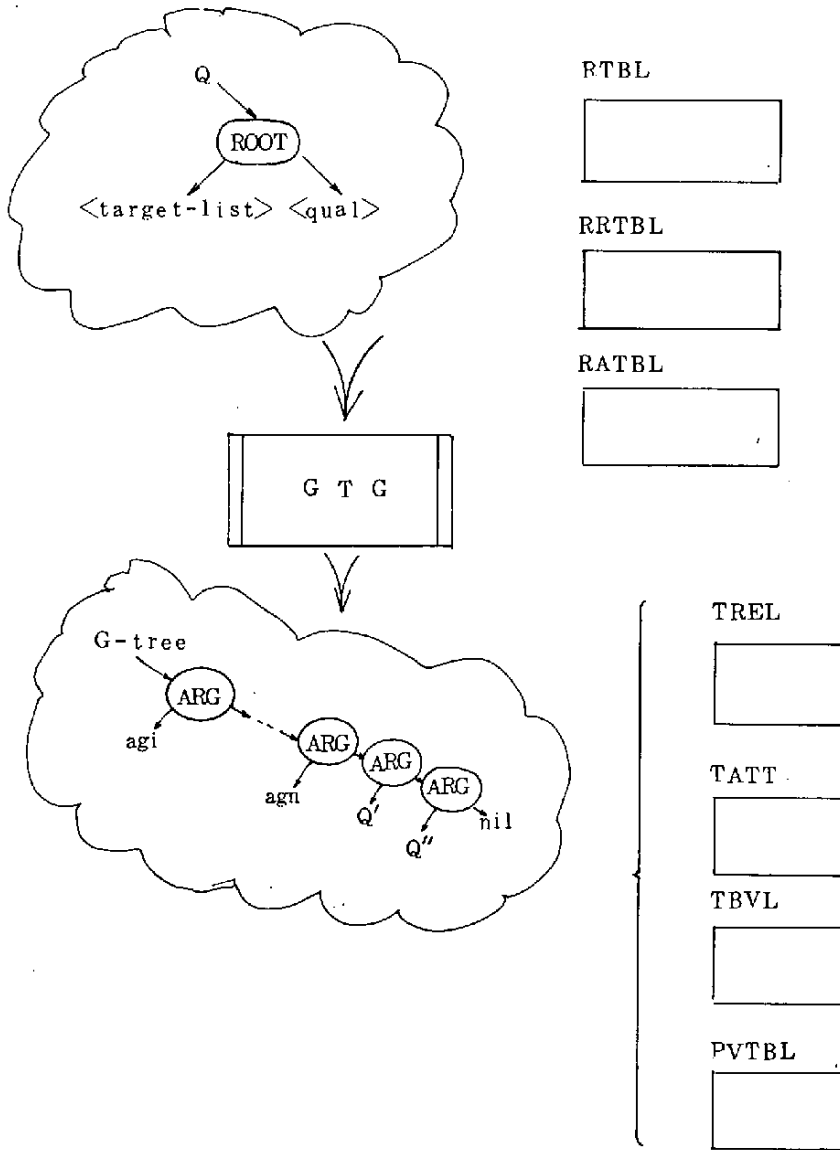


図2.13 GTG の処理概要

処理アルゴリズム

1) Q-tree の目標リストをサーチして、次の処理を行なう。

① 単純集積関数 (sa) の場合

sa に対して PV ノードを生成し、sa を PV ノードで置換する。sa は、G-tree に付加する。

② group-by 集積関数 (gf) の場合

gf を G-tree に付加する。この時、gf に対して中間結果関係名、結果属性名を生成し、TREL、TATT、RRTBL 及び RATBL に情報を格納する。

この時、group-by 変数を見つけ、group-by 変数リストを生成する。

③ Q'' の目標リストの生成

Q'' の目標リストとして Q-tree の目標リストをコピーする。

④ Q' の目標リストの生成

Q' の目標リストとして Q-tree の目標リストから gf を参照する結果属性を取り除き、sa は PV ノードで置換した目標リストを生成する。

2) Q-tree の条件式をサーチして、次の処理を行なう。

○ 単純集積関数 (sa)

sa に対して PV ノードを生成し、sa を PV ノードで置換する。sa は、G-tree に付加する。

② group-by 集積関数 (gf)

gf は G-tree に付加する。この時、gf に対して中間結果関係名、結果属性名を生成し、TREL、TATT、RRTBL 及び RATBL に情報を格納する。

この時、group-by 変数を見つけ、group-by 変数リストを生成する。

③ Q'' の条件式の生成

Q'' の条件式として、Q-tree の条件式内で gf を参照する Ci (conjunction) を見つけ、Ci からなる条件式を生成する。

④ Q' の条件式の生成

Q-tree の条件式から、gf を参照する Ci (conjunction) を除いた条件式を生成する。

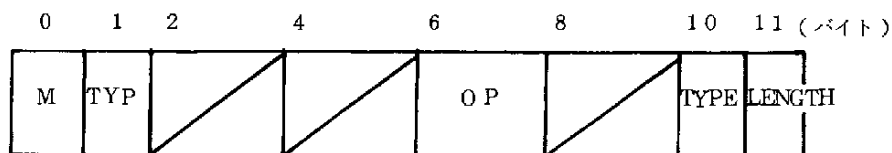
3) Q' - tree の目標リストに、group-by 変数リストの属性を付加する。

4) Q'' - tree の条件式に group-by 結合式を付加する。

Q'' - tree で参照している gf、Q' - tree の結果属性ノードを新しい属性ノードで変換する。

G-tree を構成する節点として新たに追加された節点は、Q'' 内の属性ノードと PV ノードである。

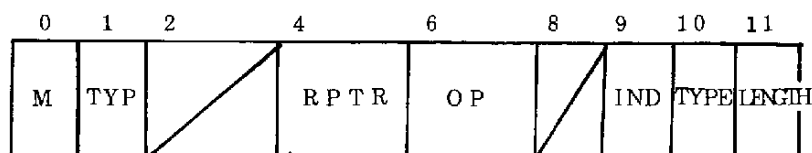
PV ノードの構成は図 2.14 の様である。



パラメーター	説 明
T Y P	TYP(P) = 3
O P	PVTBLの組番号
T Y P E	結果のデータタイプ
L E N G T H	結果のデータ長

図 2.14 PVノードの構成

Q' 内の属性ノードの構成は図 2.15 の様である。



パラメータ	説 明
T Y P	TYP(P) = 5 (ATTN)
R P T R	属性番号
I N D	K = 属性が主ソートキーの時 N = その他
T Y P E	属性のデータタイプ
L E N G T H	属性のデータ長

図 2.15 Q' 内の属性ノードの構成

表 2.13 TREL (バイト)

属 性 名	属性番号	role	type	length	説 明
GF#	1	K	I	2	集積関数番号
REL-NAME	2	N	C	6	"TEMP" gf# 2ケタ
DEGREE	3	N	I	2	
TL	4	N	P	2	gf# が生成される retrieve の目標リストへのポインタ

表2.14 TATT

属性名	属性番号	role	type	length	説明
GF#	1	K	I	2	集積関数番号
ATT-NO	2	K	I	2	属性番号
ATT-NAME	3	N	C	6	属性名
TYPE	4	N	C	1	'I' or 'C' I: 整数 C: 文字
LENGTH	5	N	I	2	バイト長
ROLE	6	N	I	1	{ K: sort-key N: otherwise
OPTR	7	N	P	2	gf#.att-no ノードへのポインタ (R.ai)
PTR	8	N	P	2	$a_i = \text{expi}$ の時 expi へのポインタ

表2.15 TBVL

属性名	属性番号	role	type	length	説明
BVL	1	N	C	4	bit-map (32ビット)
NVL	2	N	I	2	bit-map内でONとなっているbit数
QPT	3	N	P	2	treeへのポインタ
AGFLG	4	N	I	1	集積関数を含むとき '1'

表2.16 PVTBL

属性名	属性番号	role	type	length	説明
PV-NAME	1	K	C	16	'PV-' と tuple # から生成
TYPE	2	N	C	1	'C' / 'I'
LENGTH	3	N	I	2	データ・バイト長
PTR	4	N	P	2	PVノードへのポインタ

C. GDMLGの実現

GDMLGでは、G-treeから集積関数、Q'あるいはQ''に対するQ-treeを取り出してDMLを生成する。各々のQ-treeは、LDP-V2における1つの問合せと同様に処理される。Q-treeからLCS関係を参照する問合せグラフ(LQ)が生成される。LQから、構造変換によってCODASYLスキーマの要素(レコード型、セット型)を参照する非手続的問合せ(CQ)が生成される。このCQからアクセスパス(アクセス木)を生成する。そしてアクセス木からCOBOL DMLを生成する。

G D M L G の処理概要は、図 2.16 の様である。

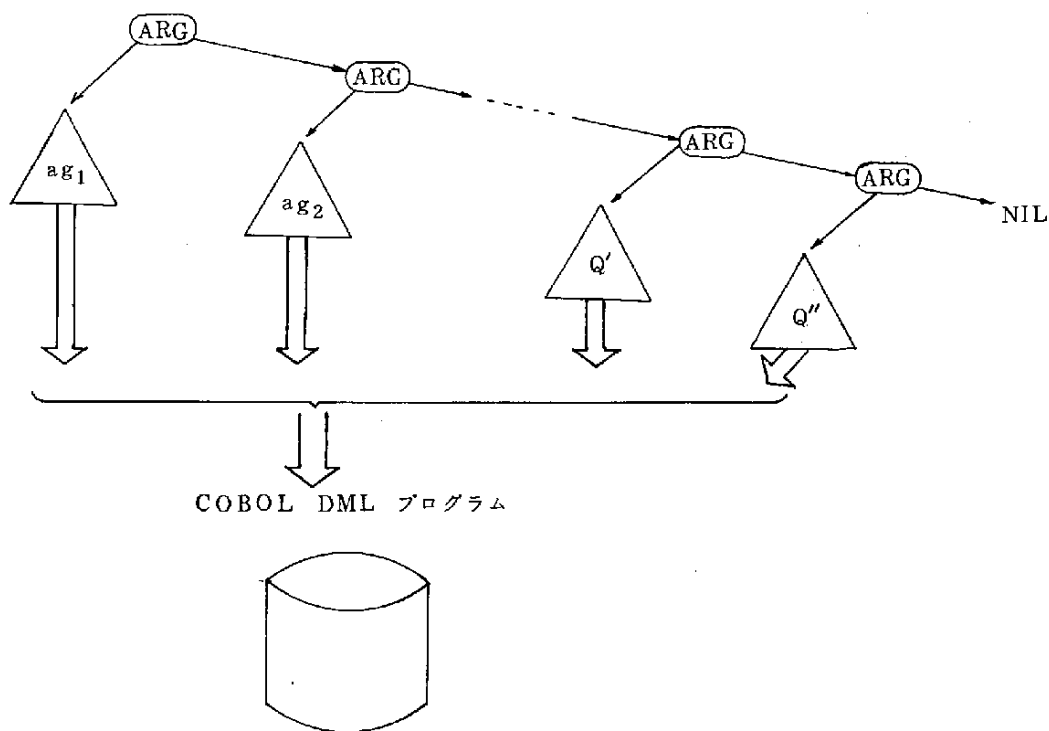


图 2.16 GDMLG 处理概要

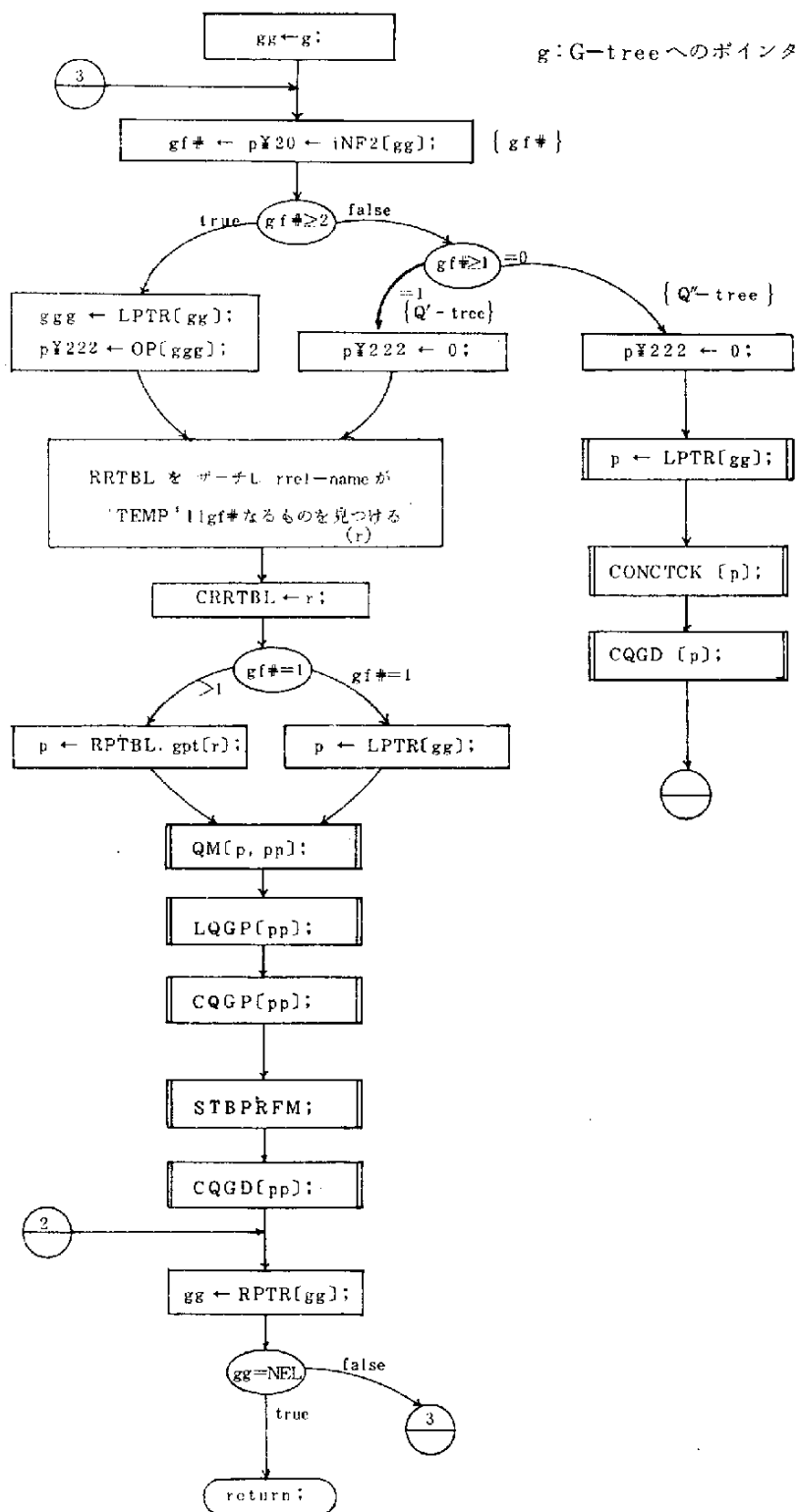


図 2.17 GDMLG 処理フロー

D. LQGPの実現

LQGPは、Q-treeと3つのテーブル(RRTBL、RATBL、RTBL)を入力して、LCS問合せグラフ(LQG)を生成する。LQGPの処理概要は、図2.18の様である。

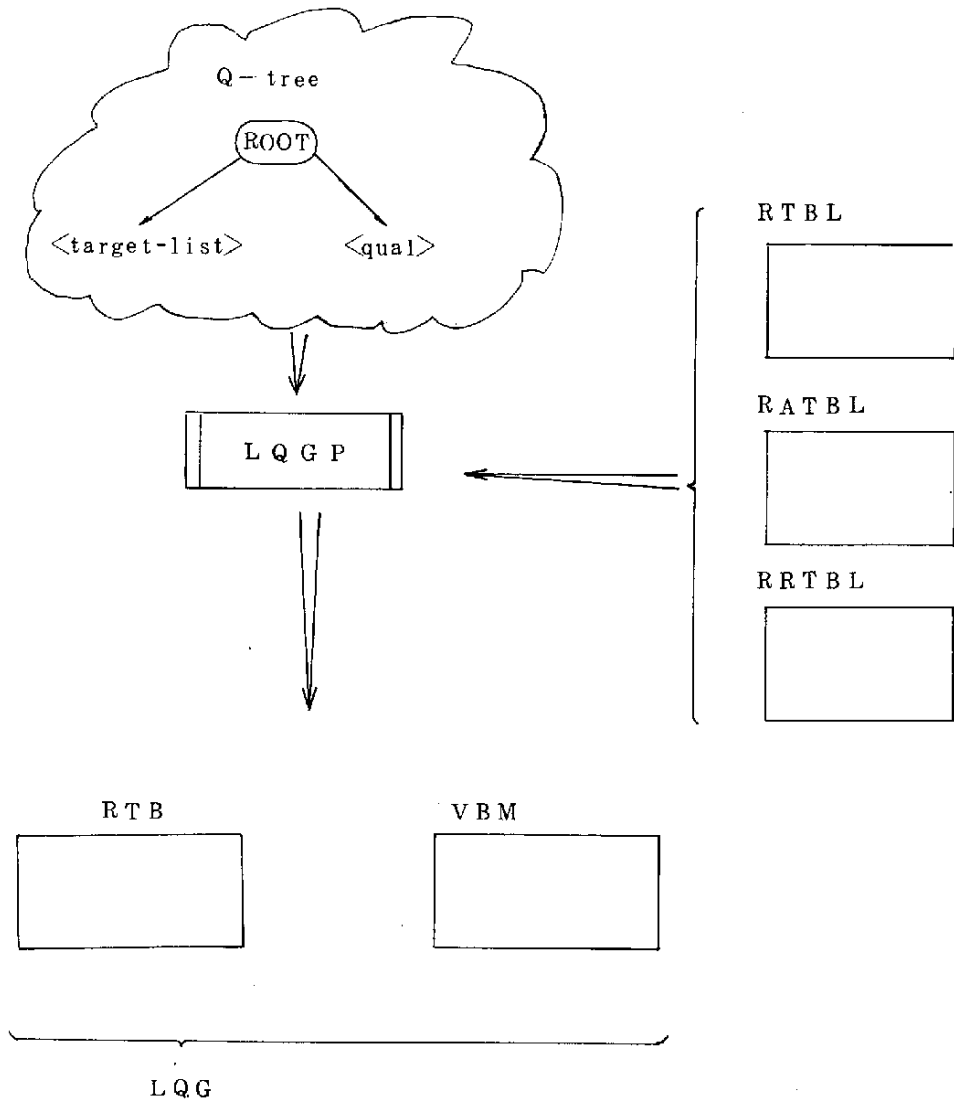


図2.18 LQGPの処理概要

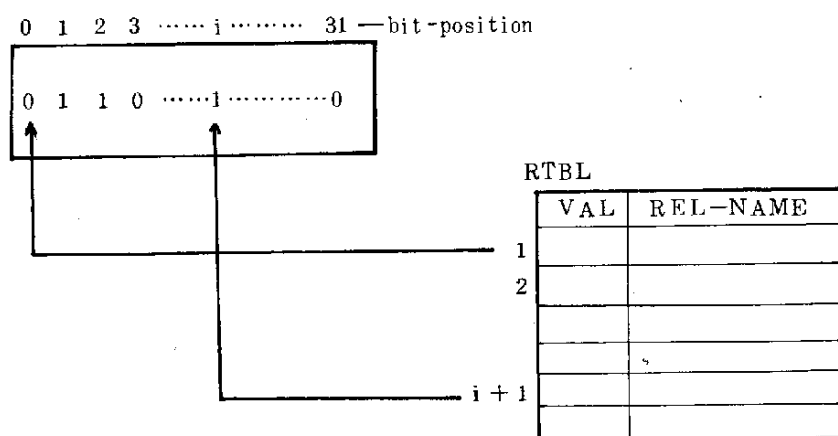
LQGは、RQL問合せの変数を節点、目標属性、条件式を辺とするグラフである。LQGは、変数及び制限式、結果属性に関する情報を格納したRTBと、変数間の結合式についての情報を格納したJBMとから成る。JBMは、ビットマップによって、どの節点についての結合式であるかを示す。

LQGPは、次の処理を行なう。

- ・問合せの参照する関係（ESR、RSR）を明らかにし、RTBにセットする。この関係についての制限と結果属性のリストをRTBにセットする。
- ・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関係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のスキームは表に示す。PTRは、目標属性又は〈conjunction〉の木へのポインタ値である。

VBLは32ビット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〈conjunct〉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のテーブル内につくられる。

RTBは、Q-tree、RTB、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関係名とこの関係に関連する情報とを格納する。

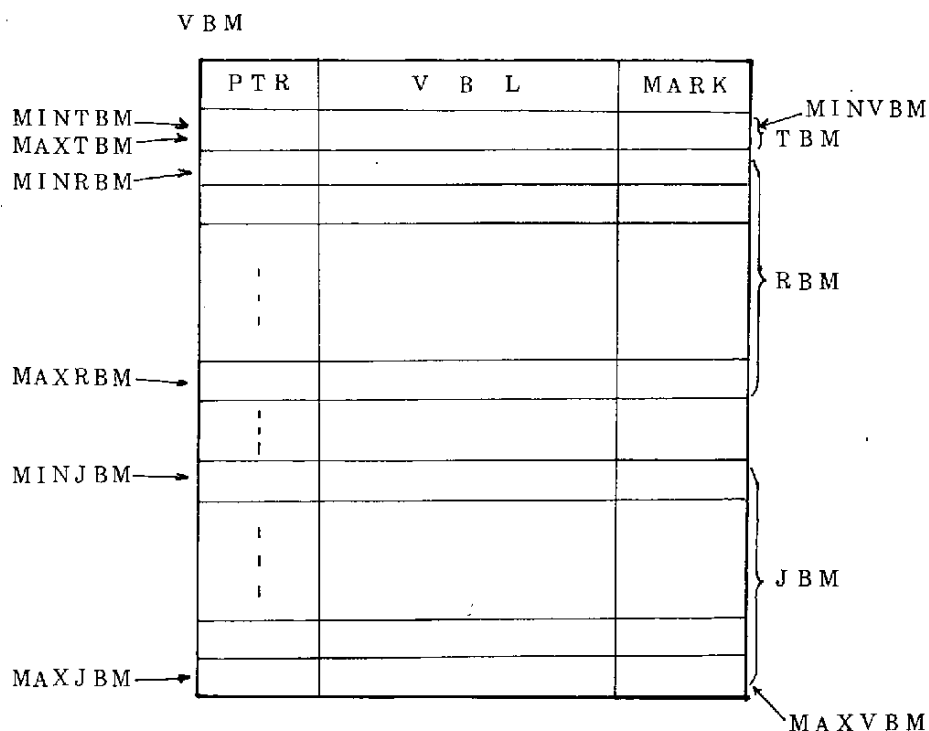


表 2.17 V B M (variable bit-map)

V B Mは、問合せの<qualification>の各<conjunct>のbit-mapを格納するための関係である。

属 性 名	属性番号	role	type	length	説 明
p t r	1	N	P	2	<conjunct>へのポインタ
t y p e	2	N	D	1	<conjunct>の式のタイプ { 等価結合 E Q J (= 1) その他 N N (= 2)
V B L	3	N	C	4	<conjunct>のbit-map
m a r k	4	N	D	1	

表 2.18 R T B (Record-type table)

R T Bは、変数に関する情報を格納するための関係である。

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	2	CALC(DNA)をもつ equi-restriction tree へのポインタ (なければNIL)
rstr	5	N	P	2	CRSTRを除いた restriction tree へのポインタ (なければNIL)
cslet	6	N	F	4	CRSTRの選択度 (CRSTR=NILなら1)
rslet	7	N	F	4	RSTRの選択度 (RSTR=NILなら1)
rslt.	8	N	P	2	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)
nptr	10	N	P	2	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	14	N	D	1	{ PRO (=3) NORM (=0) ANTI (=2) CN (=1) SN (=4)

E. CQGP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関係に、後者はSTB関係に保持される。LQGからCQGを生成する処理モジュールは、CQGPと呼ばれ、その処理概要は図2.19の様である。

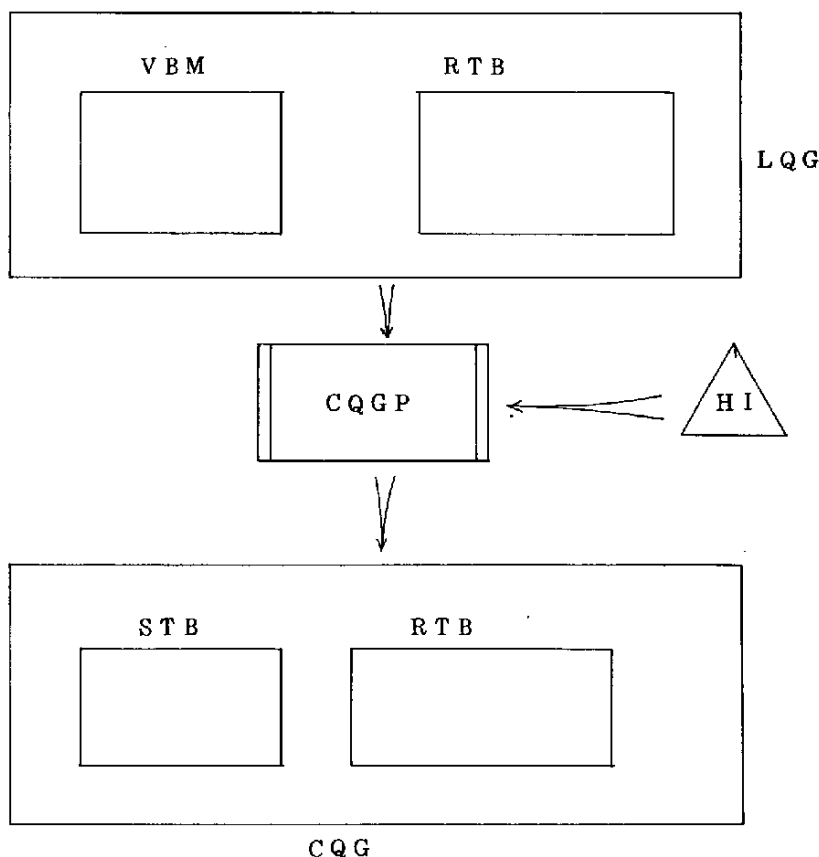


図2.19 CQGPの処理概要

CQGPは、次の処理を行なう。

i) LQG内の各 join link に対して、次の処理を行なう。

a) 主等価結合 (PEJL) の場合

(即ち、 $x.a = y.b$ $a \in pkey(x) \wedge b \in pkey(y)$)

隠れ構造を明らかにして、対応する CODASYL 構造 (セット型) を生成する。

b) 主非等価結合 (PNJL) の場合

(即ち、 $x.a \neq y.b$ $a \in pkey(x) \wedge b \in pkey(y)$)

非セットリンク (NSL) を生成する。

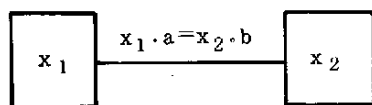
c) 非主結合 (NPJL) の場合

(即ち、 $x \cdot a \theta y \cdot b \wedge \theta \in \{<, \leq, =, \neq, \geq, >\} \wedge$
 $a \in \text{pkey}(x) \wedge b \in \text{pkey}(y)$)

非主結合リンク (NPJL) を生成する。

ii) 等価ノードを併合する。

等価ノードとは次の様な条件を満たすものである。



$\text{rel}(x_1) = \text{rel}(x_2) \wedge \text{rel-type}(x_1) = \text{rel-type}(x_2) = E \wedge$
 $a = b = \text{pkey}(x_1) = \text{pkey}(x_2)$ のとき

x_1 と x_2 は等価であるという。

等価なノードに対して次の様に併合処理を行なう。

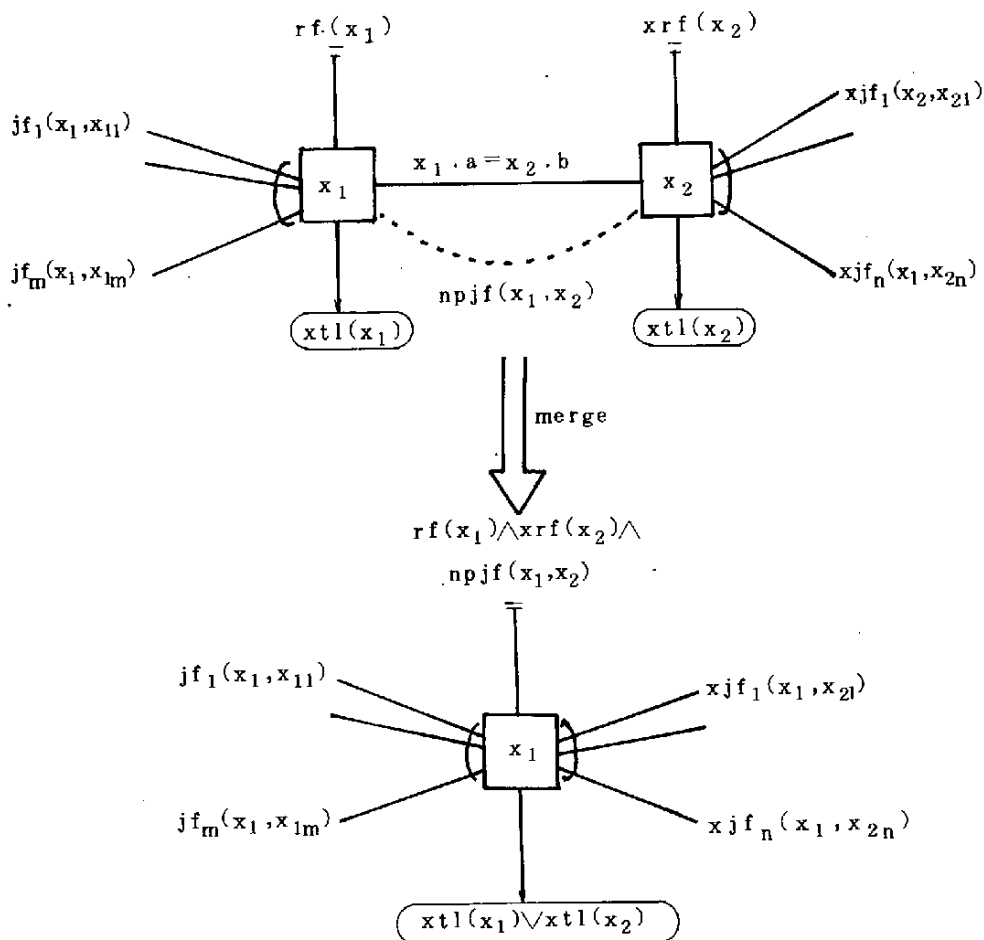


図 2.20 x_1 と x_2 の併合処理

iii) 関係モデル要素 (関係と属性) を CODASYL 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。

- ・ R 関係の属性は、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 関係) にこれを移す。
- ・ 隠れ構造が存在するため、R 関係内の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、次の様な変換方法をとる。

- ・ R 関係の属性のデータ項目への変換は、隠れ構造が全て見つけれられた時に行なう。
- ・ R 関係から E 関係への属性の移動は、属性のまま行なう。
- ・ R 関係から E 関係へ移された属性は、移された時点で、データ項目へ変換する。

表 2.19 S T B (Set-type table)

STB は CQG におけるレコード型 (ノード) 間のリンク情報を表す。

属 性 名	属性番号	role	type	length	説 明
stype	1	K	C	16	セット型名又は非セット型名又は非主結合リンク
type	2	N	D	1	ONE (=0): next link TWO (=1): next + owner NSET (=2): non-set-type
orec	3	N	D	2	owner record-type の store された RTB #
mrec	4	N	D	2	member record-type の store された RTB #
connectivity	5	N	R	2	orec から mrec への 結合度の 平均値
max-connect	6	N	R	2	orec から mrec への 結合度の 最大値
i-connect	7	N	R	2	mrec から orec への 選択度の 平均値
sort-type	8	N	D	1	N (=0) not sorted A (=1) 昇順 D (=2) 降順
sort-att #	9	N	D	2	sort key の mrec の att # (ないとき 0)

属性名	属性番号	role	type	length	説明
unique-att#	10	N	D	2	このセット内でDNA指定のされている atl-no
m-p-rstr	11	N	P	2	primary restrictionへのポインタ (なければNIL)
m-n-rstr	12	N	P	2	non-primary restrictionへのポ インタ(なければNIL)
m-p-slet	13	N	R	2	m-p-rstrの選択度
m-n-slet	14	N	R	2	m-n-rstrの選択度
p tr	15	N	P	2	join clause へのポインタ (set-type 以外のときのみセット)
insertion	16	N	P	1	{ AUTOMATIC (=0) MANUAL (=1)
retension	17	N	D	1	{ MONDATORY (=0) OPTIONAL (=1) FIXED (=2)
mark	18	N	D	2	作業用
sort-mode	19	N	D	1	primary restrictionはsort item を参照している(=1)
cop	20	N	D	1	prstrの比較演算子
dna-mode	21	N	D	1	prstrがDNA itemを参照して いる(=1)、いない(=0)
ac-direction	22	N	D	1	{ pos (=0) neg (=1) その他(=-1)

F. CQGDの実現

CQGDは、CQG(RTBとSTB)を入力して、連結な部分グラフCQG_iを生成する
とともにCQG_iの処理スケジュール(CQL、JQL、RATBM)を生成する。

CQGDの処理概要は図2.21の様である。

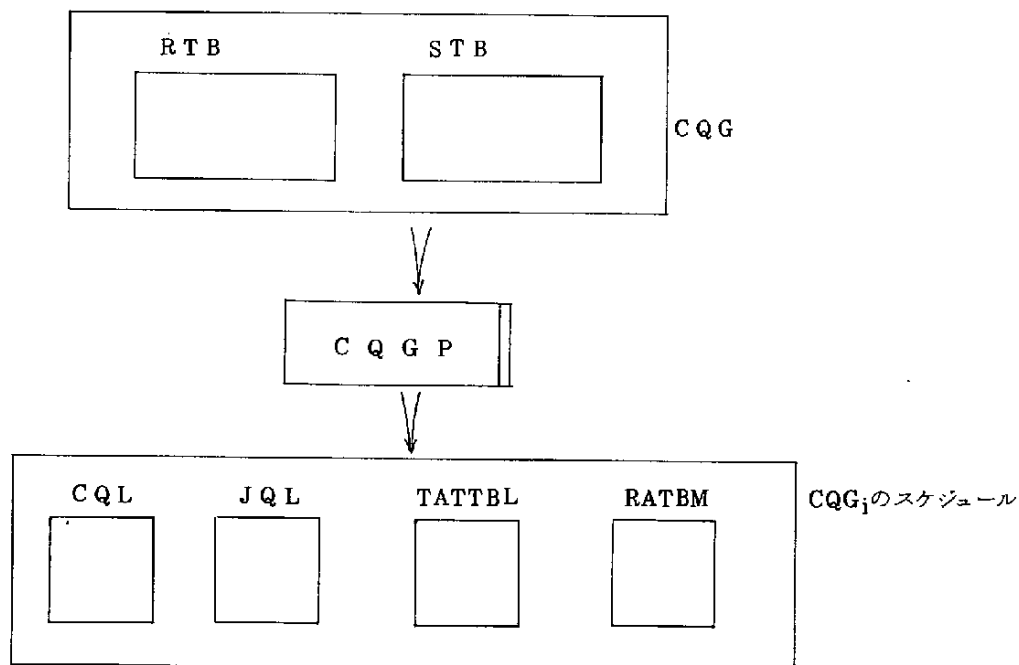


図 2.21 CQGD の処理概要

CQGD は、次の処理を行なう。

- CQG を、連結な部分グラフ CQG_i ($i = 1, \dots, n$) に分割する。
- 中間結果の結合又は、直積のスケジューリング決定する。
- Q' -tree の時、集積関数の結果と Q' -tree の結果との結合スケジュールに従って、部分グラフ CQG_i を取り出す。 CQG_i から、アクセスパスを生成し、COBOL DML を生成する。アクセスパス生成と DML 生成は、CQGD 内から call される。

表 2.20 RATBM

RATBM には、問合せの目標リスト情報がセットされる。目標リストの各要素が参照する CQG_i との対応がセットされる。但し、同一の CQG_i を参照する要素は 1 つにまとめられている。

(バイト)

属 性 名	属性番号	role	type	length	説 明
BVM	1	N	C	4 (32bits)	CQG bit-map
ratp	2	N	P	2	RAT へのポインタ

表2.21 J Q L

J Q Lは、分割されたC Q G_i間の結合式を表わしている。

属 性 名	属性番号	role	type	length	説 明
B V L	1	N	C	4 (32bits)	ビット マップ
q p t	2	N	P	2	結合式へのポインタ
mark	3	N	D	4	作 業 用
ratp	4	N	P	2	目標リストへのポインタ

表2.22 C Q L

C Q Lは、各C Q G_iの変数のビットマップ(C B M)、C Q G_iの結合リンク数などの情報を保持する。

属 性 名	属性番号	role	type	length	説 明
C Q G #	1	K	D	3	C Q G _i の i に相当
C B M	2	N	C	4 (32bits)	変数のビットマップ
connect	3	N	D	5	C Q G _i のもつ結合リンク
rel-name	4	N	C	16	中間結果関係 (T _i)
mark	5	N	D	4	作 業 用
ratp	6	N	P	2	目標リストへのポインタ
schedule#	7	N	D	3	schedule #
jqpt	8	N	P	2	出力される結合式へのポインタ
P C Q L #	9	N	D	2	schedule# + 1のC Q Lノード番号 (次のC Q G _i へのポインタ)

表2.23 TATTBL

TATTBLは、中間結果関係と、その属性、目標表現との対応表である。

属性名	属性番号	role	type	length	説明
rel-name	1	K	C	16	中間結果関係名
att-no	2	K	D	3	属性番号
att-name	3	N	C	16	属性名
gpt	4	N	P	2	目標表現へのポインタ

G. アクセスパス生成の実現

アクセスパスは、セット辺によって連結な CQG_i の節点と辺からアクセス木を生成する。

アクセスパス生成(APG)の処理概要は図2.22の様である。

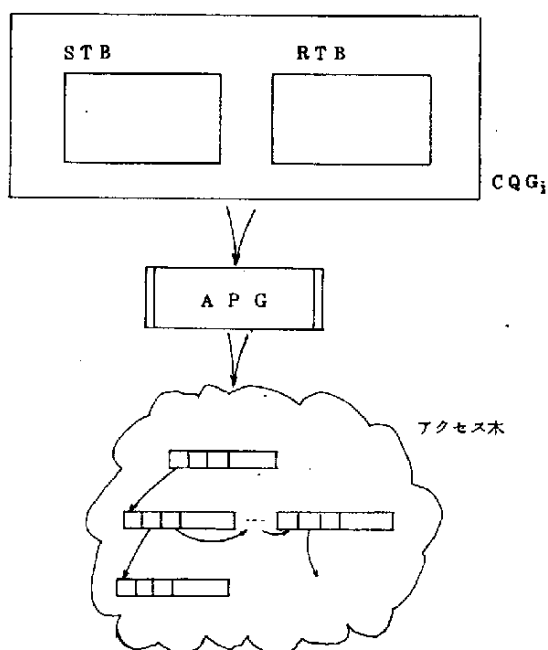


図2.22 APGの処理概要

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。

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-array木であるが、これを2分木で表わす。(図2.23)。

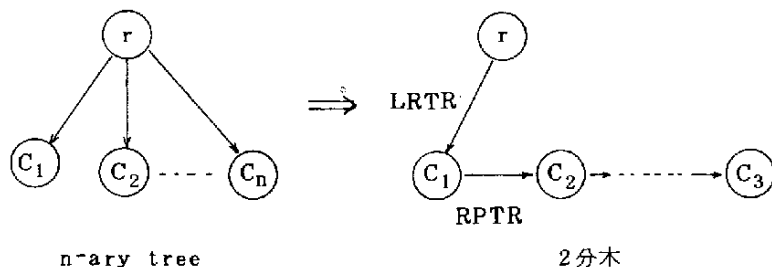


図 2.23

アクセス木節点の構成は図 2.24 の様である。アクセスパスは、DFA アルゴリズムによって生成される。

0	1	2	4	6	7	8	9	10	11 (バイト)
M	TYP	LPTR	RPTR	REC	SET	STYP	NTYP	NODE-NO	

パラメータ	説 明
TYP	AT 節点を示す。(= 10)
LPTR	子節点の先頭節点へのポインタ
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	NORM (= 0) normal 節点 CN (= 1) 合流節点 ANTI (= 2) 反節点 PRO (= 3) pro 節点 SN (= 4) synonym 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

図 2.24 AT 節点の構成

H. DML生成の実現

DML生成 (DMLG) は、プログラムの共通部分や各集積関数、 Q' 、及び Q'' に対するCOBOL DMLを生成する。各 $CQGi$ から生成されたアクセス木(AT)を入力して、COBOL DMLを生成する。DMLGの処理概要は図2.25の様である。

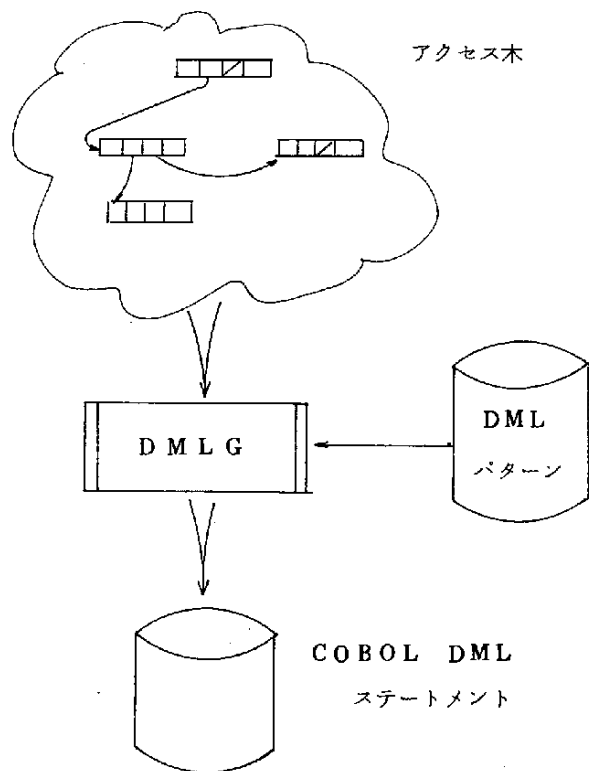


図2.25 DMLGの処理概要

DMLGは、パラメータDMLSTUSの値によって生成するDMLパターンが異なる。DMLSTUSの値は、CQGDでスケジュールに従ってセットされる。各DMLSTUSの値に対して生成されるDMLパターンは表2.24の様である。DMLパターンの詳細は、付記4 DMLIを参照されたい。

表 2.24 DML生成パターン

DMLSTATUS	処 理	生成パターン
0	各 gf, Q', Q'' の共通部分のDML生成	ID-S DATA-FR DATA-S DATA-WS 5 DATA-WS 6 DATA-WS 4
	各 CQG_i 内のノードに対するアクセスパターンの生成	<u>アクセスパターン</u> { FIRST : NEXT
1	各 CQG_i 内のノードに対するアクセスパターンの生成	<u>アクセスパターン</u> { FIRST : NEXT
9	各 gf, Q' の結果生成処理パターンの生成	RREL 2 NODUP <u>集積処理パターン</u> { SUM : MAX
99	各 gf_i と Q' の結合処理DMLパターンの生成。結合処理の初期処理パターン(RREL)と生成。	RREL { RREL 3 RREL 4 (any, count の時は RREL 4を生成)
100	各 gf_i と Q' の結合処理DMLパターンの生成	RREL 3 RREL 4
999	プログラム共通部分のDMLの生成。	ID-P DATA-P DML-INI DATA-WS STOP FRTRN SUB
	更新処理パターンの生成	<u>更新パターン</u> { APP-L APP-E : DEL-E

2.3.5 結果関係の管理

利用者の実行した問合せに対する結果関係はRWSに作られ、管理される。

結果関係に対して、結果関係のスキーム（結果関係名及び属性構成）はLDPが生成する。又LDPから起動されたCOBOL DMLが問合せを実行し、その結果をRWSに格納する。結果関係の組をRWSに格納する処理モジュールLOADWSは、LDPから起動されるCOBOL DML実行ジョブ内の1つのジョブステップである。

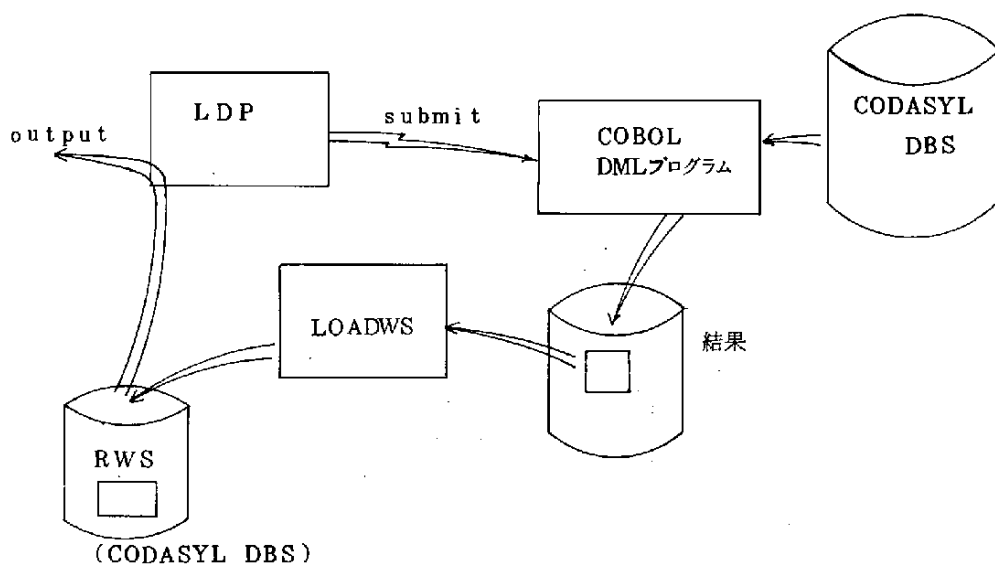


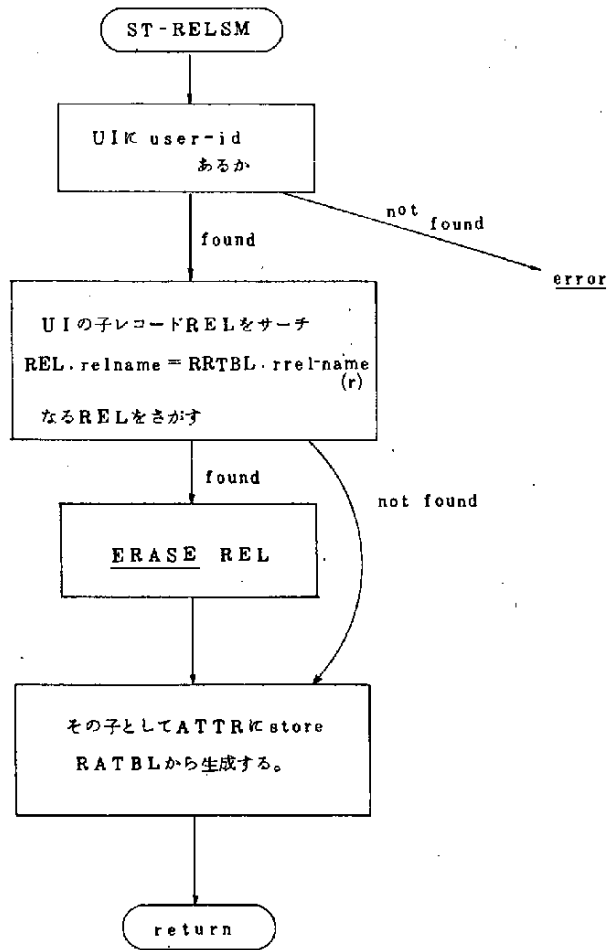
図 2.2 6

A. 結果リレーションのスキーマの生成

```
procedure ST-RELSM(r,user-id);  
input : r=RRTBL番号  
       user-id
```

処理概要

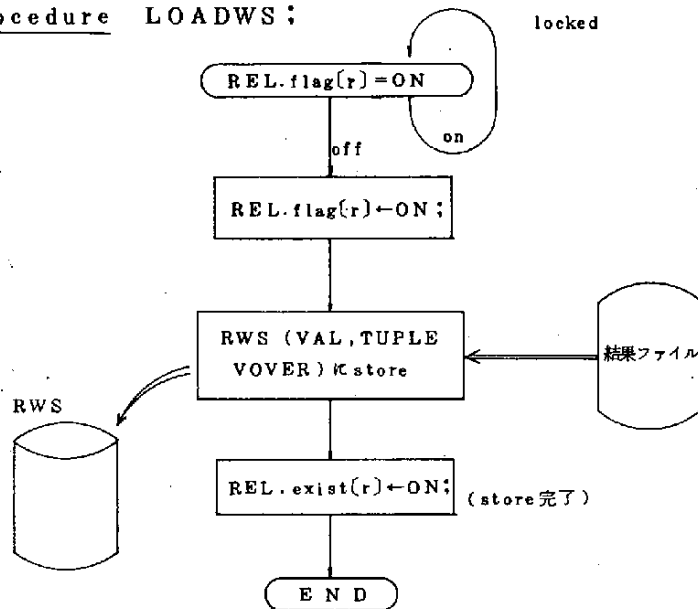
- 1) user-idをCALC項目としてUIをサーチする。
- 2) UIのmember record-type RELをサーチしてrelname=RRTBL.rrel-name{r}を見つける。
- 3) 即ちRELにあればERASEする。
- 4) RELにstore
- 5) RELのmemberとしてRATBLからATTRをstoreする。

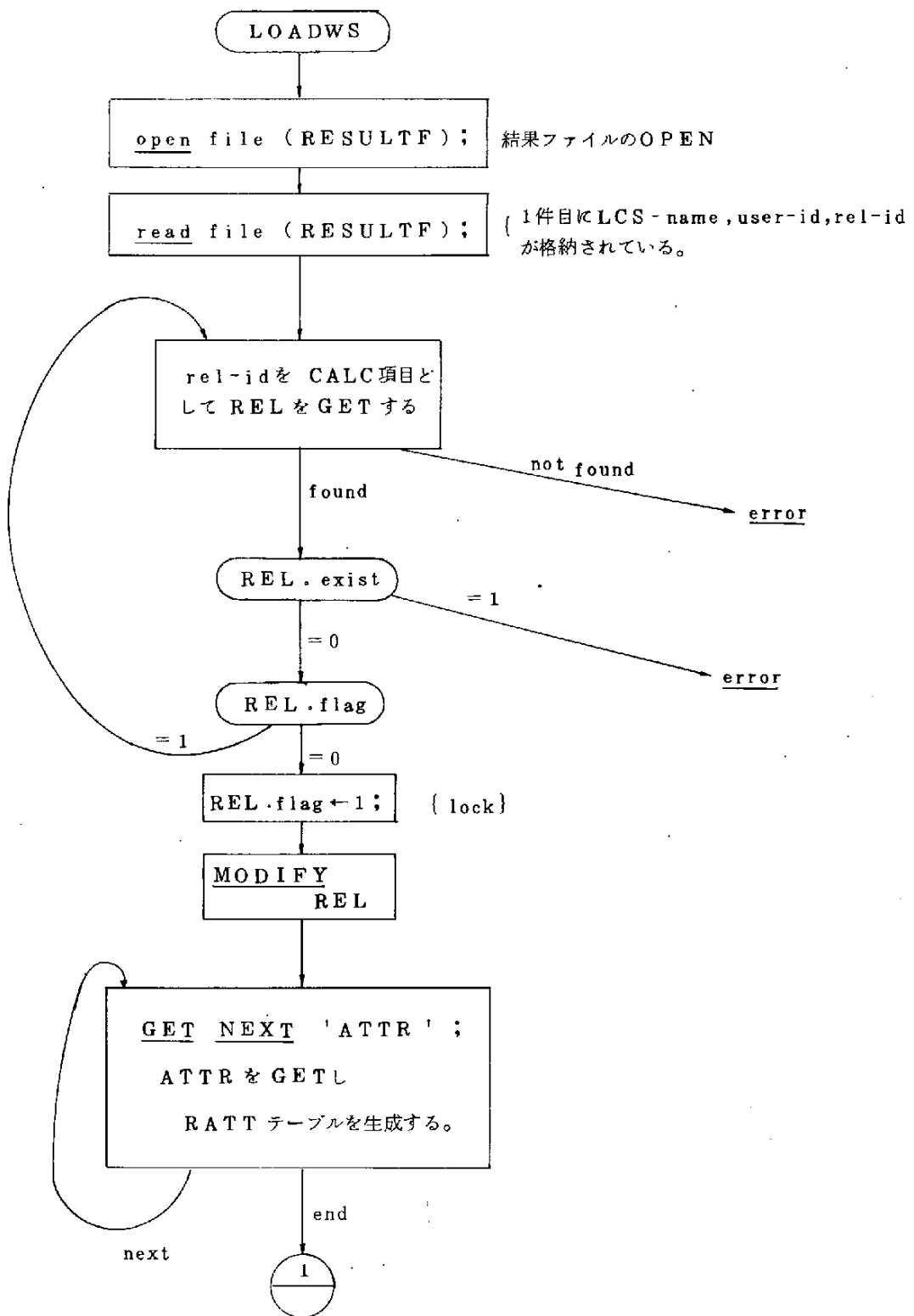


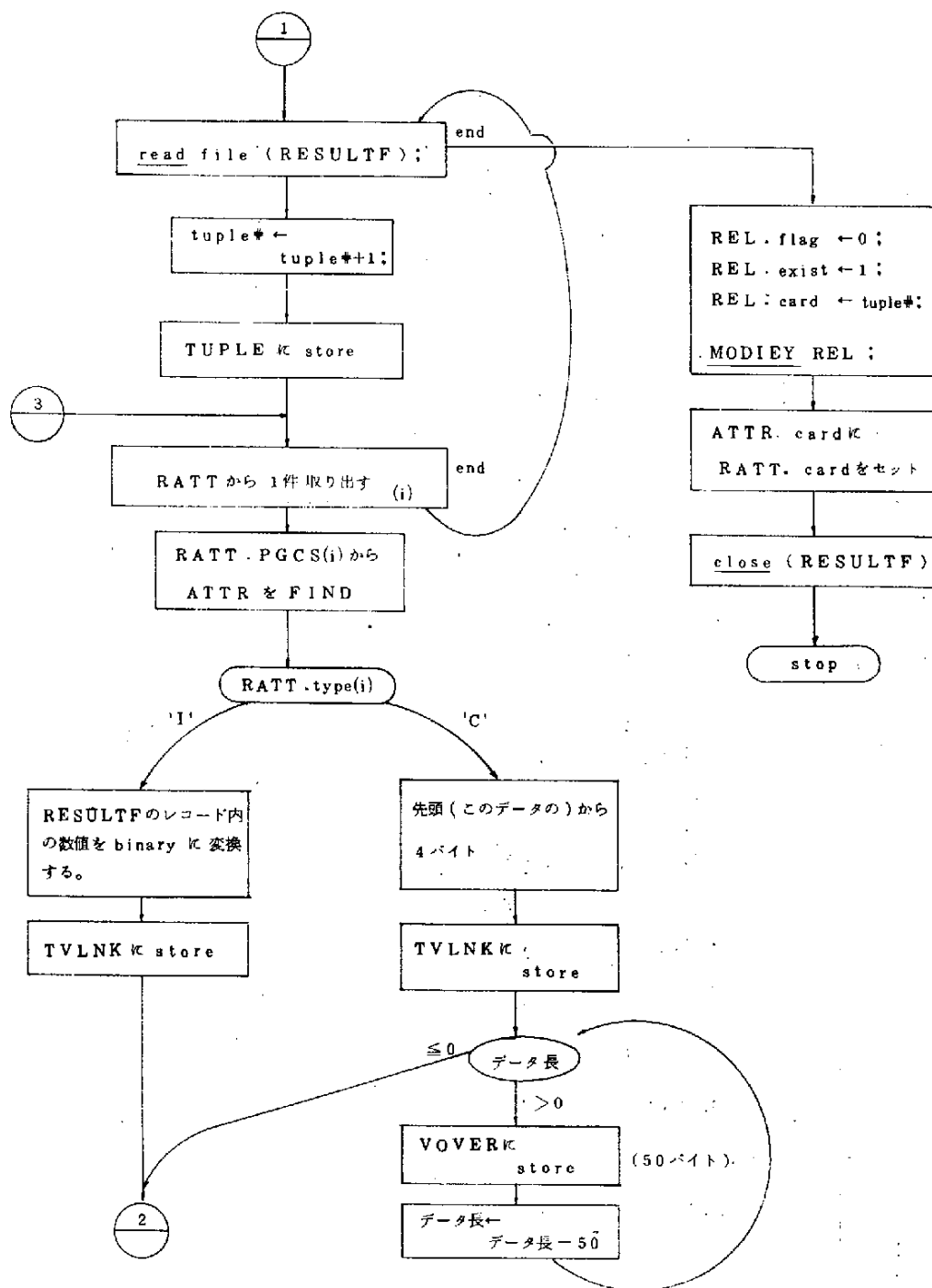
B. 検索結果の格納

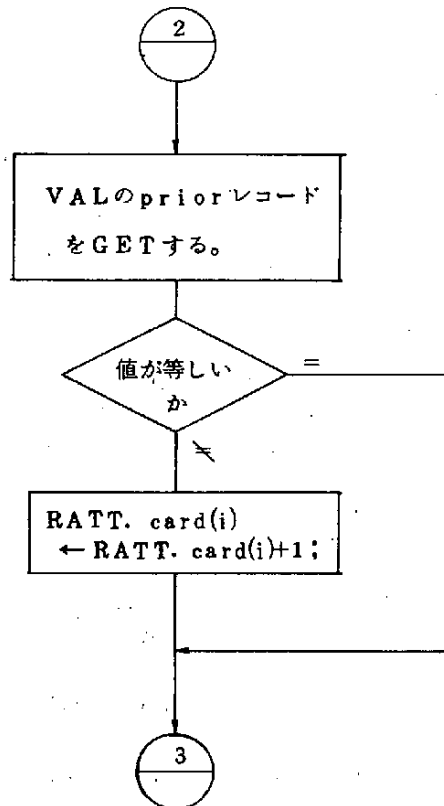
procedure LOADWS;

概要









2.3.6 COBOL DML の起動方法

QTPで生成されたCOBOL DMLは、バッチジョブとして起動される。COBOL DMLに対するジョブ制御文は、QTPによって生成される。COBOL DMLがアクセスするDBSに対するサブスキーマ名、PED名が生成される。これらは、QTP実行時に指定するLCS名によって決定される。

COBOL DMLの起動方法としては、次の2つの方法がある。

1) QTPがバッチジョブとして起動された場合

この場合、COBOL DMLの起動方法は、インタナルリーダーによって起動する。

2) QTPがTSSのもとで実行された場合

この場合、IPFコマンドを使用してCOBOL DMLジョブをsubmitする。

2.3.7 LDPの同時実行制御

LDPの利用者が問合せを実行させ、その結果を得る為には、LDPとCOBOL DMLプログラムを同時に実行させることが必要である。

LDPに対して次の様な仮定を設ける。

- LDPは単一利用者のもとで実行される。
- LDPで同時に実行されるCOBOL DMLプログラムは唯一つである。

上記の仮定のもとで、同時実行制御を行なう問題は、次の3つのパターンになる。

- i) LDPとCOBOL DML プログラムとの同時実行制御
- ii) 複数LDP間の同時実行制御
- iii) COBOL DML プログラム間の同時実行制御（複数LDPが各々1つのCOBOL DMLプログラムを実行。）

各々のパターンにおいて共有されるリソースは次の様である。

- i) LDPとCOBOL DML プログラム

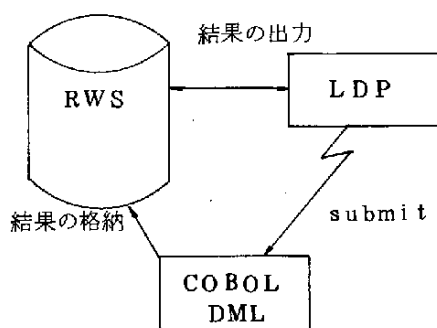


図2.27 LDPとCOBOL DMLの共有リソース

LDPとCOBOL DML プログラムは関係作業領域（RWS）を共有する。

- ii) LDPとLDP

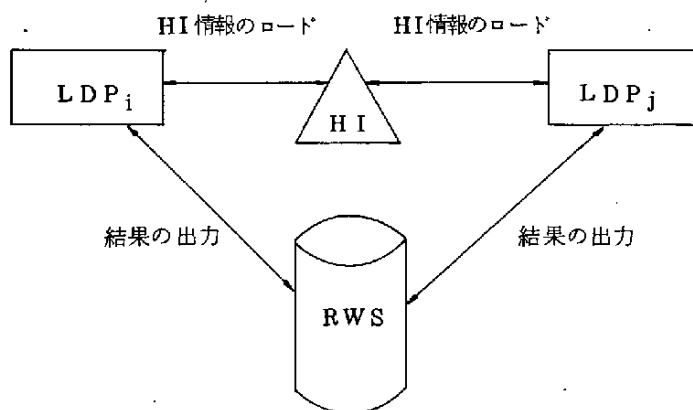


図2.28 LDP間の共有リソース

LDP間ではHI 情報とRWSを共有する。

- iii) COBOL DML とCOBOL DML

COBOL DML 間では、問合せに対する検索あるいは更新を行なうDBS（e.g PRD BS又はDCDBS）とRWSを共有する。

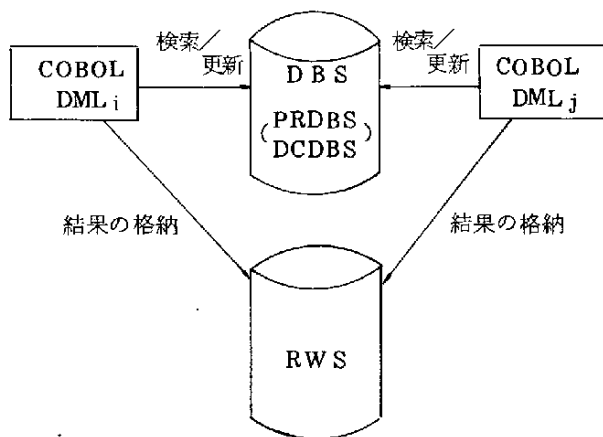


図2.29 COBOL DML間の共有リソース

A. AIMの排他制御

AIMでは、トランザクション単位でデータベースの管理を行なう。トランザクションは、あるデータの処理の範囲に対して設定される。トランザクションは、システムによる設定とアプリケーション内での `transaction start/end` 命令 (PL/Iでは、`PLITRNS`, `PLITRNE` の `call` 命令) による設定とがある。AIMは、このトランザクション単位にデータベースの排他制御を行なう。

応用プログラムの実行時のPED文で、共有又は非共有モードの実行力が指定される。

B. LDPとCOBOL DML間の同時実行制御

LDPとCOBOL DML間ではRWSを共有する。LDPとCOBOL DMLの同時実行制御を次の様に行なう。

- i) LDPがRWSに結果関係スキームを格納する。この時、RELレコード型内の状況コードを実行中コードにセットする。
- ii) LDPがCOBOL DML 実行ジョブを起動する。
- iii) COBOL DML 実行ジョブでは、結果をRWSに格納する。格納が終了した後、RELレコード型の状況コードに終了コードをセットする。
- iv) LDPでは、RELレコード型内の状況コードを判定して、利用者に結果を出力する。

C. LDP間の同時実行制御

LDP間では、HIとRWSを共有する。HIに関しては、LDPが起動された時にアクセスし、主記憶内にロードする。起動後は、主記憶内に格納されたHI情報を用いて処理を行なう。

RWSに関しては、結果関係の管理を利用者毎に行なう。結果関係の識別子内に利用者識別子が含まれる。結果関係はその識別子をCALC項目とするRELレコード型のもとに管理される。

D. COBOL DML間の同時実行制御

COBOL DML間では、DBSへのアクセスとRWSへの結果の格納処理に対して同時実

行制御を行なうことが必要である。

DBSへのアクセスでは、実行されるCOBOL DMLが更新を行なう場合、他のCOBOL DMLと同時に実行させない。検索を行なうCOBOL DMLの場合は、他のCOBOL DMLと同時に実行させる。この為、LDPではCOBOL DML 実行ジョブに対して次の処理を行なう。

- i) 更新問合せのCOBOL DML実行ジョブのPED文に対して非共有モードのPED文を生成する。
- ii) 検索問合せのCOBOL DML実行ジョブのPED文に対しては共有モードのPED文を生成する。

RWSへの結果の格納処理では、トランザクションの範囲を小さくして、同時実行を可能にする。トランザクションの単位は、結果関係の組を格納する処理とする。

3. 全体データベースプロセッサ (GDP)

3.1 全体データベースプロセッサ (GDP)

全体データベースプロセッサ (GDP) は分散型データシステム (DDBS) の分散型問題の解決を行なうソフトウェアシステムであり、各データモジュール毎に存在する。

3.1.1 機能概要

GDPの機能としては、全体管理者 (GA) 向けのGCS管理機能とGSC利用者向けのGCS問合せの分散処理機能 (分散問合せ処理機能) とがある。

(1) GCS管理機能 (DIP) GA用

DIを管理するためのGCS管理機能としては、以下のものがある。

- (i) 全体管理者 (GA) に対してLCS情報 (LCS関係スキーム、存在DM、etc) を表示する。
- (ii) GAの記述したGCS定義式 (GCSD) から、GCS情報、GCS-LCS対応情報を生成して、DIに格納する。
- (iii) GCSの保守 (GCS定義の修正、GCSの消去) を行なう。
- (iv) GCS利用者の管理 (定義、修正、消去)
- (v) DIへの直接アクセスを、GAがRQLを用いて行う。
- (vi) DIのパフォーマンス情報の更新 (e.g. 各DM内のデータベースの変化に対応したPIの統計情報の更新が必要になる。)

(2) 分散問合せ処理機能 (DQP) GCS利用者用

利用者の入力したGCSに基づき、RQLで書かれた問合せ) を、各DMで分散実行させて結果を利用者に出力する。

(i) GCS情報 (スキーム) の表示——display

(ii) [表現交換]

GCS問合せからLCS関係を参照する全体LCS問合せ (GLQ) に表現変化 (ET: expression transformation) を行う。

(iii) [問合せ分割]

GLQから各DMで独立に、処理出来る部分を分割する。GLQは各DM内で独立に処理出来るLCS問合せ (LQ) の集合と、DM間で処理する必要のあるDM間問合せ (DMQ) とに分割する。

(iv) [初期ローカル問合せ処理 (ILQP)]

分割されたLQを各DMに実行させる。各DMにLQをLDPでCOBOL DMLに変換し、CODASYL DBS上で実行させ、結果をRWS (関係作業領域) に格納する。

(v) [DM間通信処理]

ILQP後DMQを、GDP間での通信処理によって処理させ解を得る。

(vi) [Q & A 機能]

1つの問合せ結果に対する繰り返し問合せを許す。例えば、まずヒット件数だけを表示するなど。

(vii) [視野サポート]

GCS上の利用者視野 (EXS) の定義、修正、消去と視野に対するアクセスを許す。

(viii) GLQ (全体 LCS 問い合わせ) の利用者インターフェース。関係の所在を意識して、問い合わせを記述する。

(ix) RWS (relational working storage) の管理作業用関係の生成、消滅、演算を行う。

3.1.2 GDP の構成

GDPは図 3.1 に示す様な4つの処理モジュールから構成されている。

(1) DIP (distribution information processor)

(2) DI (distribution information)

(3) DQP (distributed query processor)

(4) RWS (relational working storage)

DIPは、GDPのGCS管理機能を行なう。DIPは、GA (global administrator) のコマンドに基づいて、分散情報 (DI) の管理を行う。

インタフェース①はGAへのインタフェースである。DIPコマンドがこのインタフェースになる。

インタフェース②は、DIPとDIとのインタフェースである。ここではDIをCODASYL DBS (具体的にはAIM) を用いて実現している。この為、インタフェースはPL/I DML又はCOBOL DMLとなる。

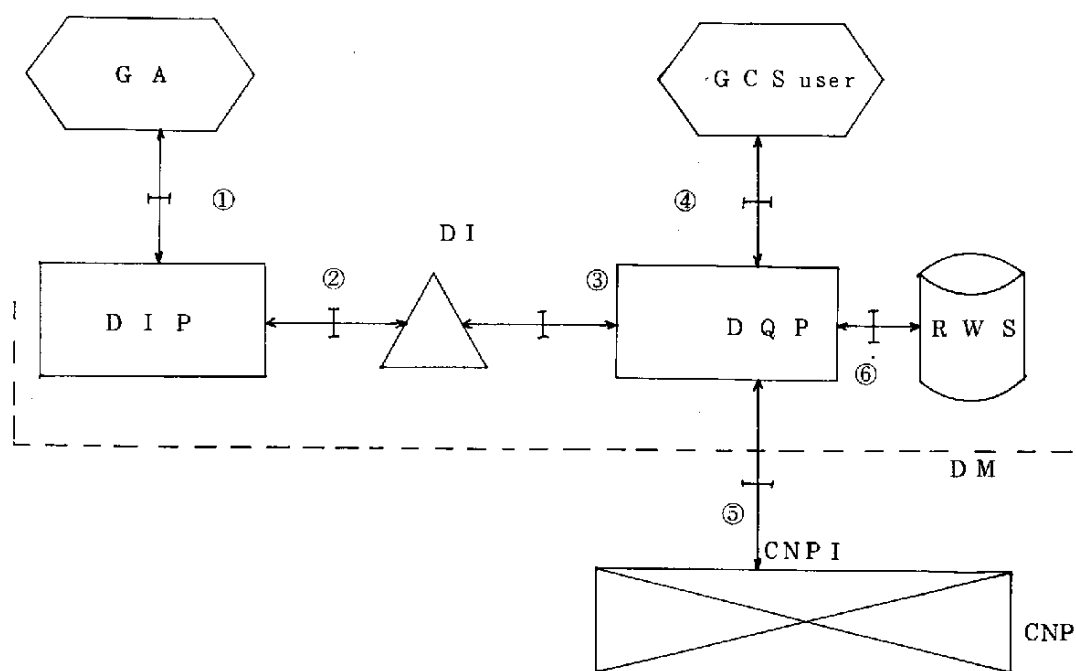
DIは、DD/Dとして、GCSの記述、GCSとLCSとの対応情報、LCS情報を有している。

DQPは、分散問合せ処理機能を行う。DQPは、GCS利用者の入力したGCS問合せを処理して、他のDMとの通信処理によって結果を得る。

RWSは、この通信処理のための作業エリアであり、中間に関係が格納され関係演算される。

インタフェース④は、利用者インタフェースで利用者に対して、RQL言語とGCSとが提供される。RWSもCODASYL DBS (AIM) で実現されるので、インタフェース⑥はPL/I DMLとなる。

インタフェース⑤は、他のDM内のDQPとの通信処理用インタフェースである。



- ①
インターフェース番号 — I —
- ① DIP インタフェース
 - ② PL/I DML
 - ③ PL/I DML ^{or} (COBOL DML)
 - ④ DQP インタフェース (RQL)
 - ⑤ CNPI コマンド
 (RQL
関係
制御コマンド)
 - ⑥ PL/I DML

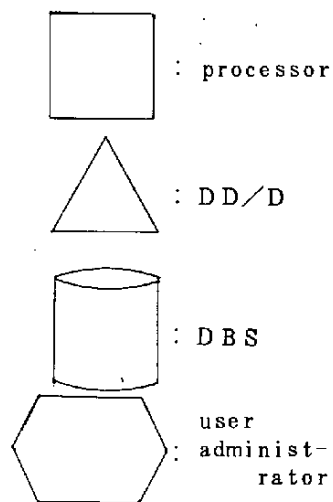


図 3.1 GDPの概要

3.1.3 GDPコマンドクラス

GDPへのコマンドは、次の4つのクラスからなる。

- (1) 第1 class 状態0から1への遷移を行う〔付記2〕。

login GDP <user-id>/<password>; GDP session の開設を行う。

- (2) 第2 class 状態1から2ケタの状態への遷移を行う。

i) DQP <gcs-name>; DQPへのアクセス

ii) define <gcs-name>; <gcs>の定義

iii) update <gcs-name>; <gcs>の更新

iv) destroy<gcs-name>; <gcs>の消去

v) logout; (RWSのclear)

GDP sessionを閉じる。

- (3) 第3 class 2ケタの状態から3ケタの状態への遷移を行う。

i-1) <DQP-command>; GCS問合せの入力。

i-2) Vdefine <view-schema>; 視野の定義を行う。

i-3) update <view-schema>; 視野の更新を行う。

i-4) destroy <view-schema>; 視野の消去。

i-5) VQP <view-schema>; 視野へのマクセスを行う。

ii-i) <DIP-Command> GCSの管理を行う。

iii-i)

- (4) 第4 class 3ケタの状態内の遷移を行う。

i-2))-1 <VIP-command> 視野の管理を行う。
i-3)

3.2 DIPの実現

3.2.1 DIPの概要

DIPは、GA(global administrator)のコマンドに基づいて、分散情報(DI)の管理を行なう。

インタフェース①は、GAのインタフェースである。DIPコマンドがこのインタフェースとなる。

インタフェース②は、DIPとDIとのインタフェースである。DIは、CODASYL DBS(e.g. AIM)を用いて実現している。この為、このインタフェースはPL/I DMLとなる。

DIはDD/Dとして、GCSの記述、GCSとLCSとの対応情報、LCS情報を有している。

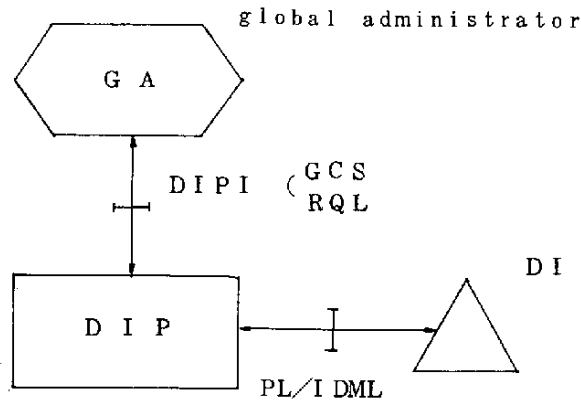


図 3.2 DIP の概要

3.2.2 DI の定義と実現

A 定 義

分散情報 (DI) は、GCS 自身の記述と GCS との層間対応情報である。

DI は、次の情報から成っている。

- (1) GCS の記述 [GCS 情報 (GCSI)]
- (2) GCS と LCS との対応情報 [GCS-LCS 対応情報]
- (3) LCS の情報 [LCS 情報]
- (4) GCS 利用者情報 [利用者定義]

B DI の表現

分散情報 (DI) は、CODASYL DBS (AIM) によって実現した。DI の CO-DAYL スキーマを図 3.2 に示す。DI は、各 DM で全く完全冗長に保持される。

(1) GCS の記述

GCS 情報は、GCS 名を CALC 項目とする GCS レコード型とセット型でリンクされた GATT、GDEF レコード型に格納される。

(2) GCS と LCS との対応情報

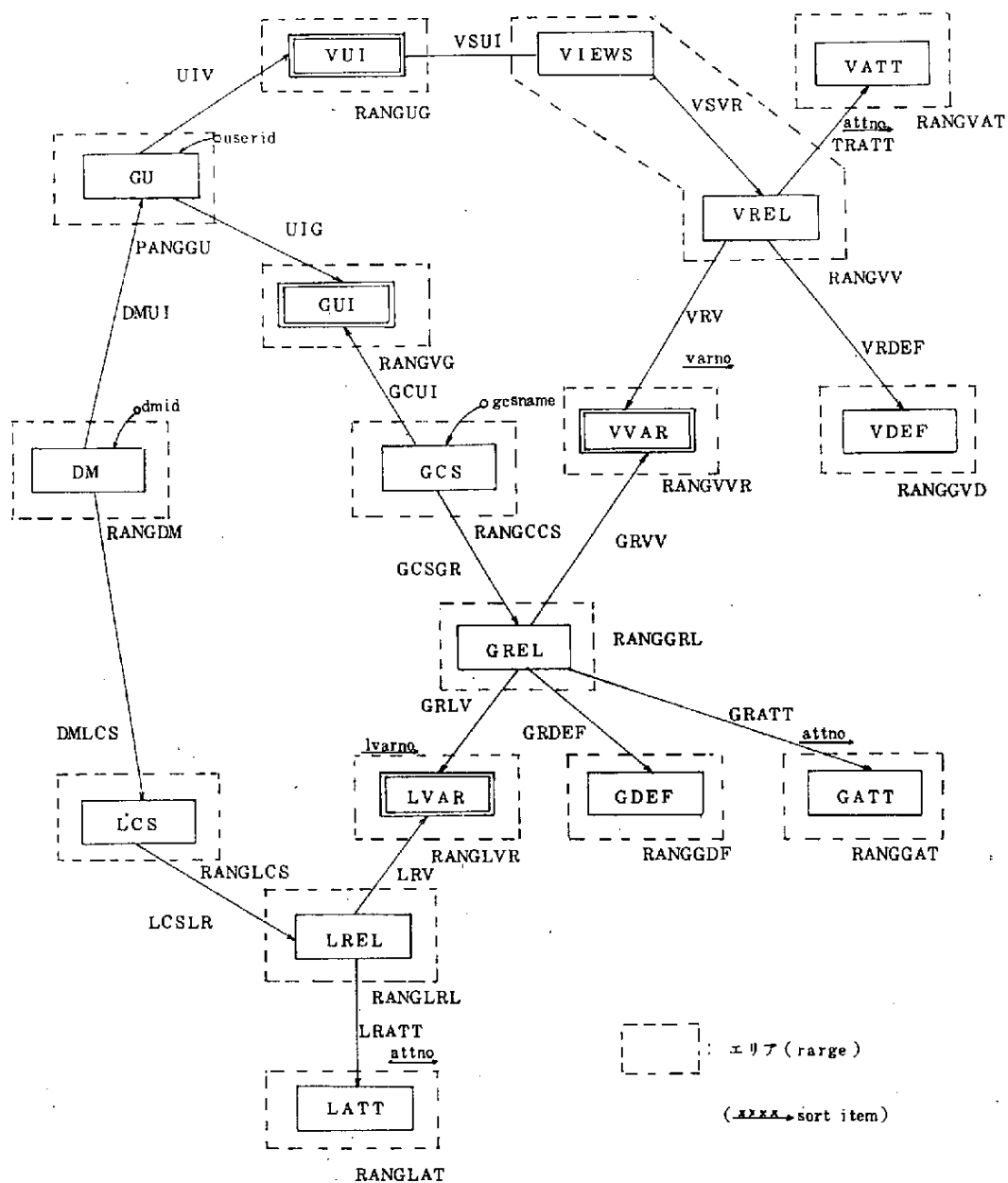
GCS と LCS との対応情報は、LVAR レコード型に格納される。

(3) LCS の情報

LCS の情報は、LCS、LREL、LATT レコード型に格納される。

(4) GCS 利用者情報

GCS 利用者情報は GU に格納され、利用権が与えられた GCS との関係を GUI リンク レコードとして格納する。



C. DIPインタフェース

DIPインタフェースは、GCS管理者(GA)の為のものである。GAに対して次の様なDIPコマンドが提供される。

(1) 新GCSの定義

```
login GDP <user-id> / <password>;
```

GDPへのsessionを開設する。

```
/9999 define <gcs-name>;
```

新たに<gcs-name>をGCS定義することを、宣言する。この時<user-id>は、このGCSの管理者となる。

```
#9999 <DIP-command>;  
: :  
#9999 END;
```

*

*は繰り返し入力される。ここでGCS関係の定義、GCSユーザの管理を行う。

<DIP-command>の詳細は、付記2 GDPコマンドを参照されたい。

(2) GCSの消去

```
login GDP <user-id> / <password>;
```

```
/9999 destroy <gcs-name>;
```

GCS<gcs-name>の消去を行なう。

(3) GCSの修正

```
login GDP <user-id> / <password>;
```

```
/9999 update <gcs-name>;
```

<gcs-name>の定義式を修正する為、必要な情報をDIからロードする。

```
#9999 <DIP-command>;  
: :  
#9999 END;
```

繰り返し

3.2.3 GCS定義式の内部表現変換アルゴリズム

DIPコマンドで定義されたGCSの定義式は、CODASYL DBSで実現されたDIに格納される。この時、GCS定義式は内部表現(コード化)に変換されてDIに格納される。

GCS定義式は、次の様にして内部表現に変換され、DIに格納される。

1) GCS定義式から、GCS定義treeを生成する。

DIPコマンドによるGCS定義は、次の様である。

```
range ( $\ell_1, L_1 : S_1$ ) ..... ( $\ell_m, L_m : S_m$ );
```

```
define G ( $ga_1/GA_1/f_1$  .....  $ga_h/GA_h/f_h$ )
```

$C_1 \cup C_2 \cup \dots \cup C_n$;

$C_i ::= C_{i1} \cap C_{i2} \cap \dots \cap C_{i\ell_i}$ (for $i=1, 2, \dots, n$)

$$C_{ij} ::= (ga_1 = gexp_{ij1}, \dots, ga_h = ga_h = gexp_{ijh}) \\ \text{where } gqual_{ij}$$

このGCS定義式に対して、次の様なtreeを生成する。

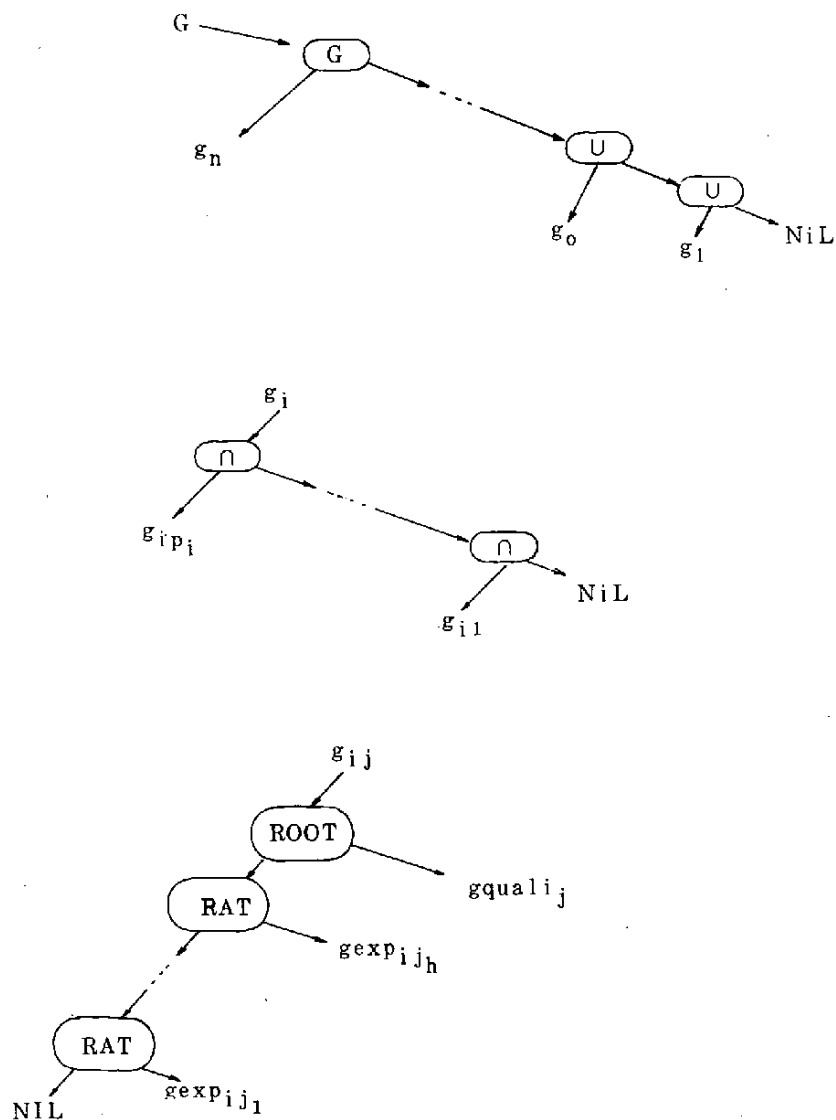


図 3.4 GCS 定義式に対する木

2) GCS定義木から、DIの情報を生成する。

DI内のGCS定義式を格納するCODASYLスキームは、図3.5の様である。GCS名の情報は、GCSレコード型に、GCS関係の情報はGRELレコード型に、GCS関係の属性構成情報はGRATTレコード型に格納される(図3.6)。

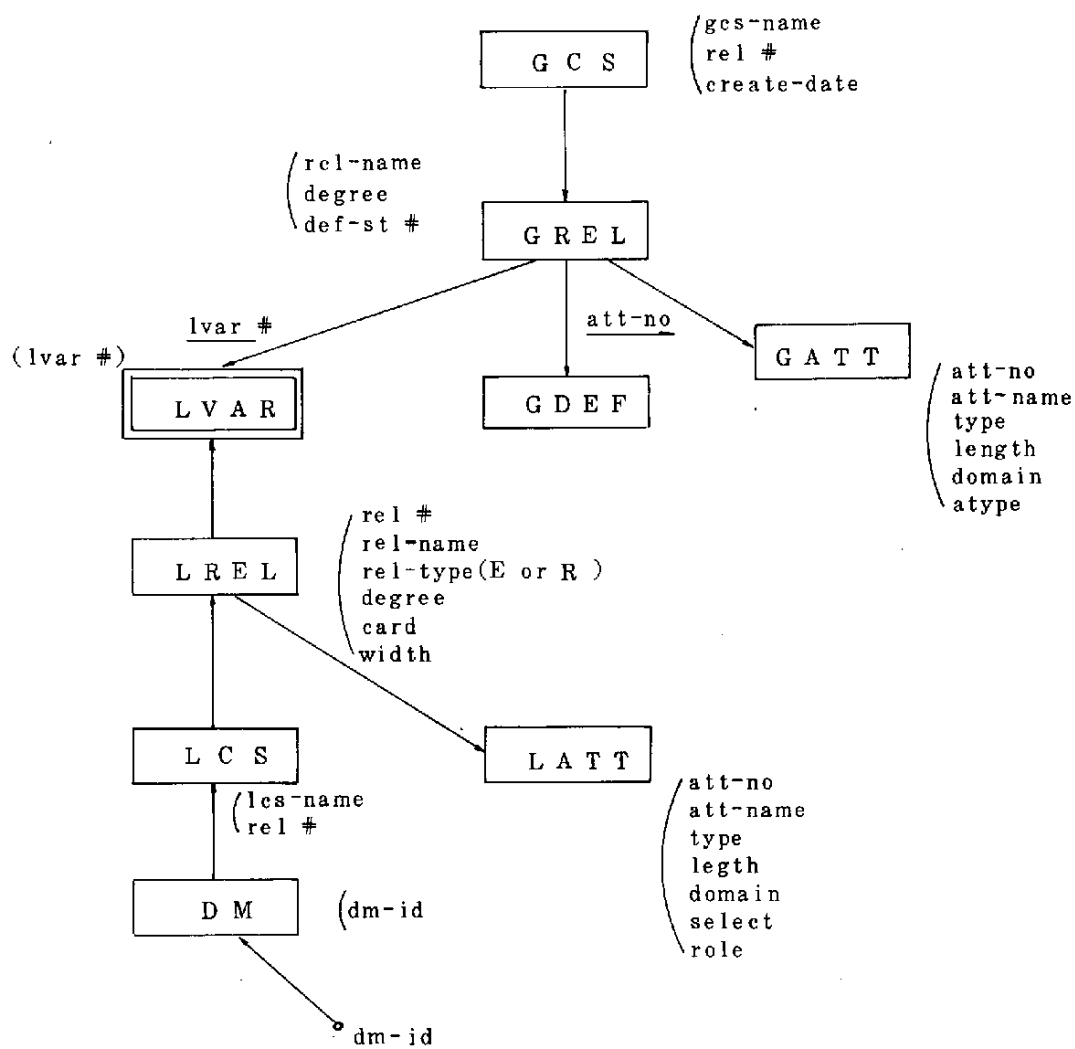


図 3.5 DI スキーム (一部)

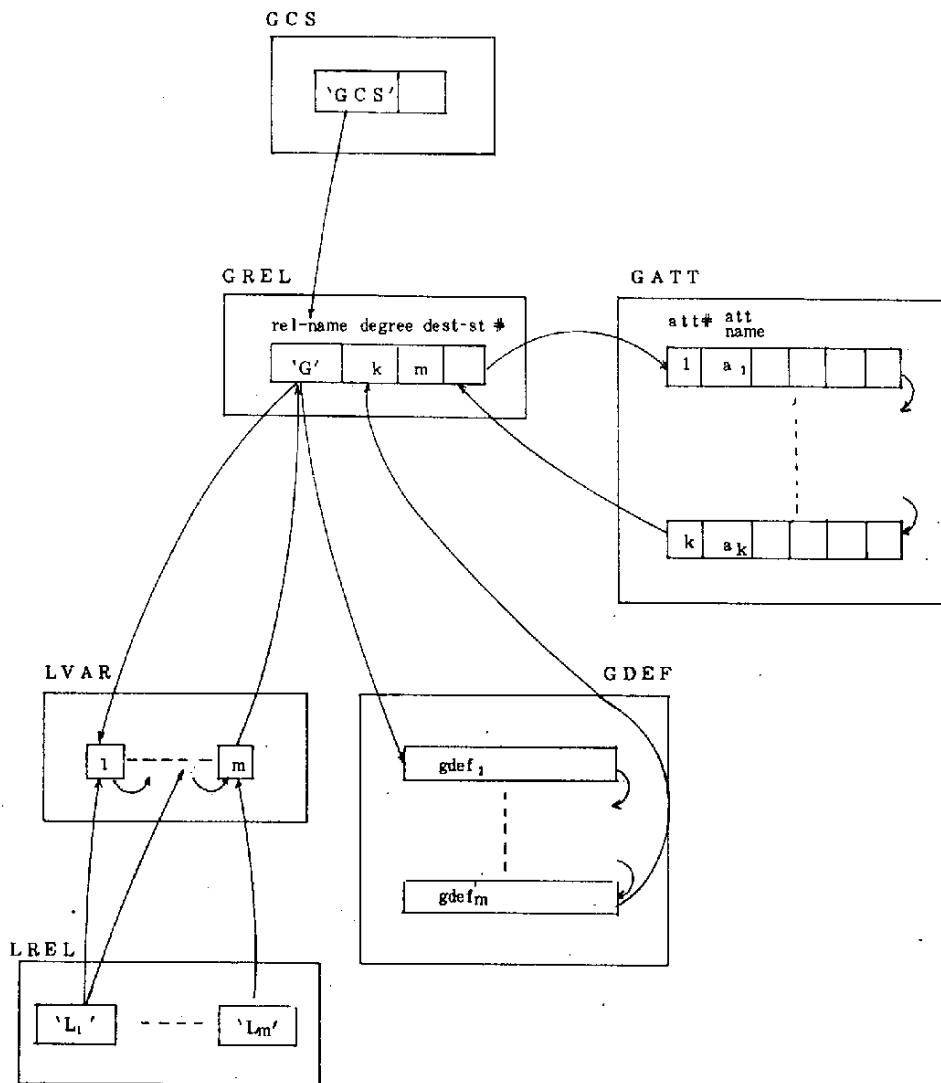


図 3.6 DI の 実 現 値 例

3) GCS定義式の内部表現変換

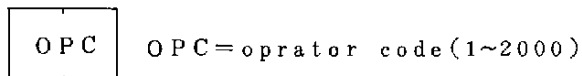
GCS定義式<g-def> は、GDEFレコード型内に code 化した内部表現<g-def-c>として格納される。<g-def-c>は次の様に定義される。

```

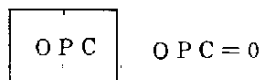
<g-def-c>      ::= <g-def-m> <g-def-term>
<g-def-m>      ::= <gd-unit> | <gd-unit> <g-def-m>
<gd-unit>      ::= <control-unit> | <data-unit>
<control-unit> ::= <nil>          | <operator>   | <vatt>   |
                  <integer-c> | <character-c> | <g-def-term>

```

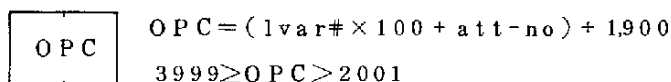
(1) operator node



(2) NIL node

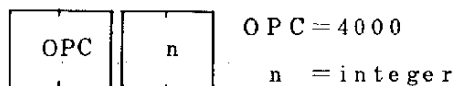


(3) vatt node

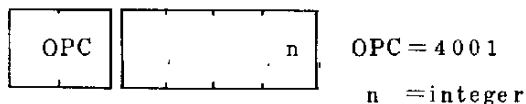


(4) inter node

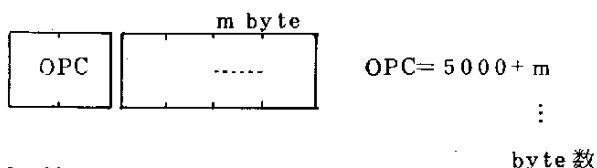
embedded integer



integer



character



(5) g-def-term node



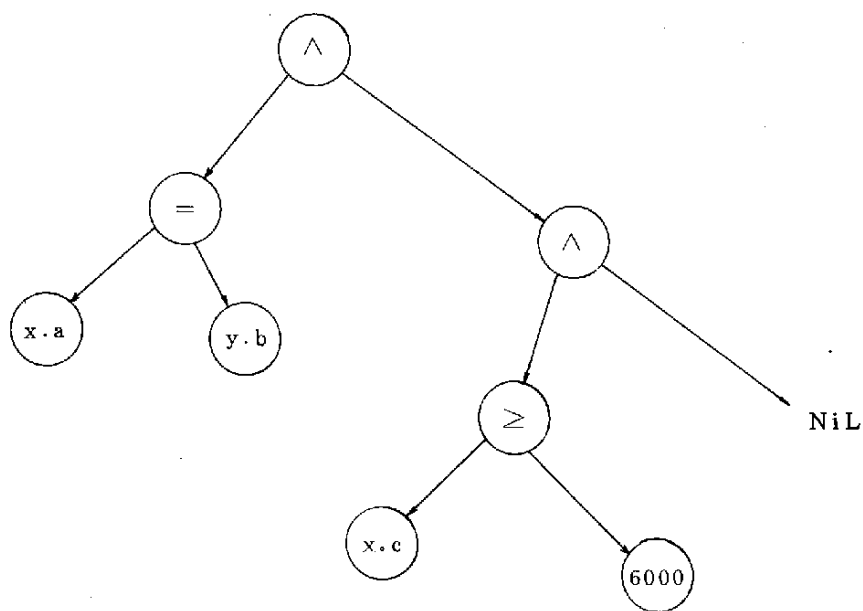
図 3.7

GCS定義tree <g-def>から、<g-def-c>は、post-fix notationとして生成される。

<g-def>から、<g-def-c>の生成例を図3.8に示す。

GCS定義が、DQPで参照される場合、<g-def-c>からGCS定義式を生成する。

<g-def>から、<g-def-c>を生成する処理モジュールはQTRIEXPとして実現した。又、<g-def-c>から<g-def>を生成する処理モジュールはQITREEとして実現した。処理モジュールの概要を次の様である。



<g-def>



内部表現への変換

<g-def-c> =

x.a	y.b	=	x.c	4000	6000	≥	0	^	^
-----	-----	---	-----	------	------	---	---	---	---

NIL

図 3.8 <g-def-c> 生成例

procedure QTRIEXP[p, buff, j, ℓ]

input : p = tree へのポインタ

buff = コード化した列を格納する領域

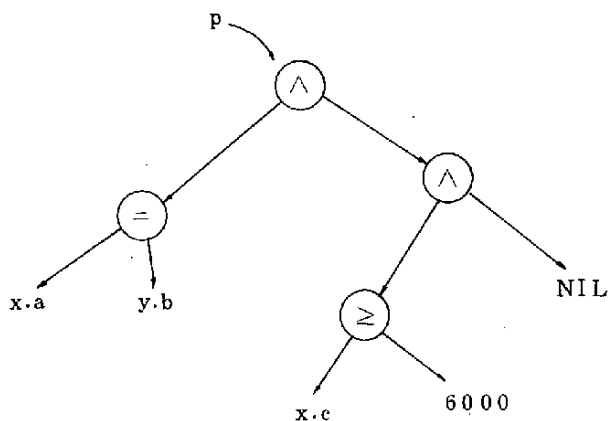
j = "

の先頭ポインタ

output j = コード化した列の最終ポインタ + 1

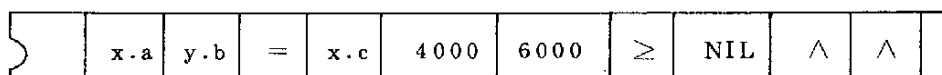
ℓ = コード化した列のバイト長

処理概要



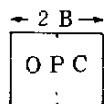
buff.

j'



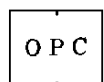
$\ell = 22$ [バイト]

1) operator node



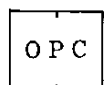
OPC = operator code (1~2000)

2) NIL



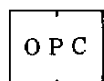
OPC = 0

3) attribute node



OPC = $\left(\frac{\text{RPTR} * 100 + \text{OP}}{\text{lvar \#} \quad \text{att\#}} \right) + 1900$

4) PV node

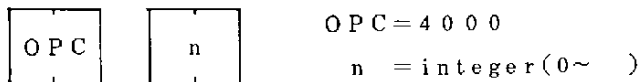


OPC = 6000 + OP

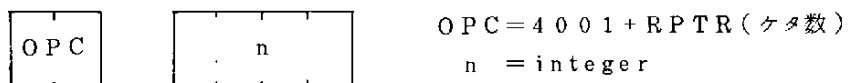
 PVNTBL#

5) constant node

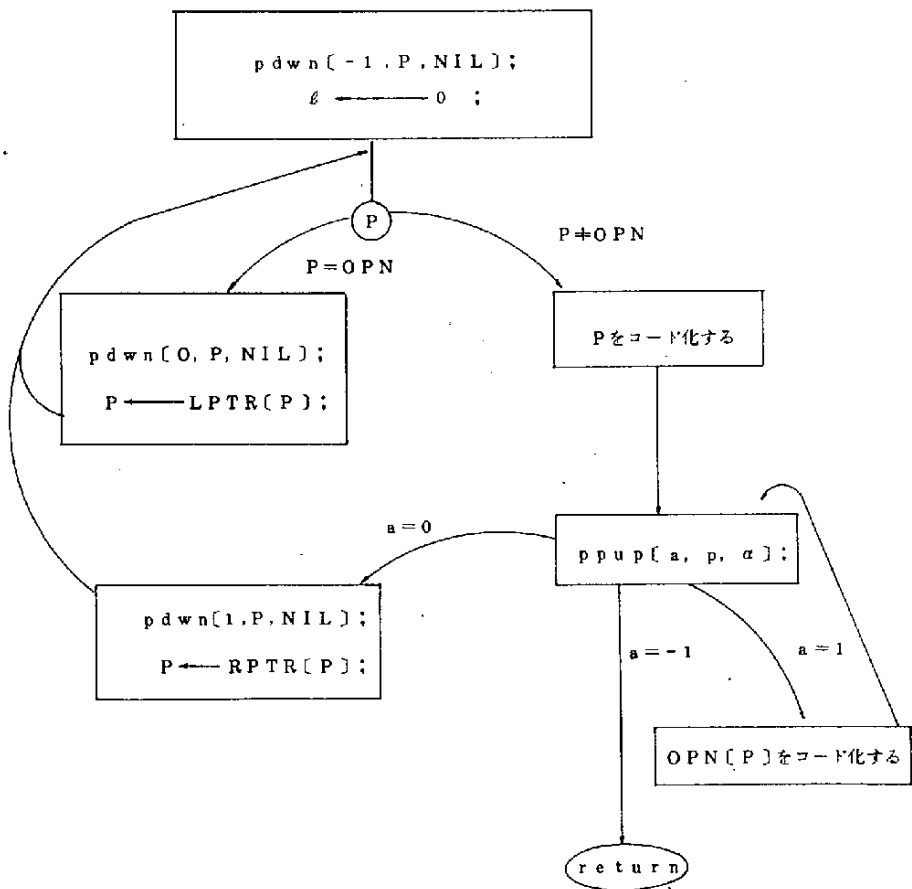
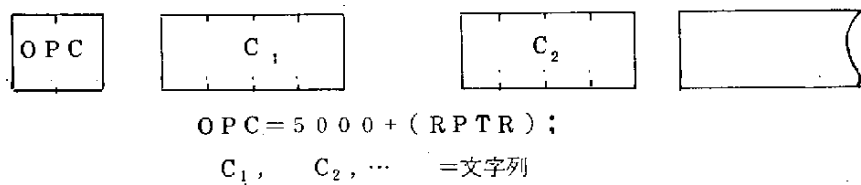
embedded integer



integer



character



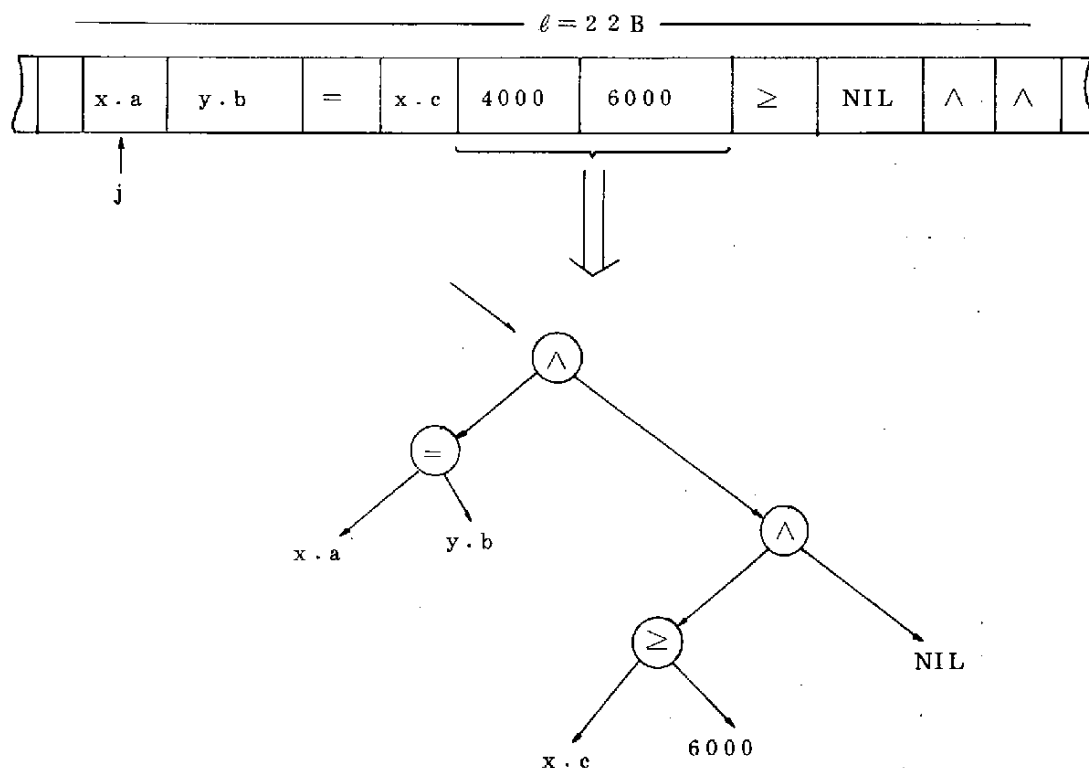
procedure QITREE[buff, j, ℓ] ;

input : buff = コード化された内部表現の格納されているバッファ
j = buff 内の先頭ポインタ
ℓ = 内部表現のバイト長

output : QITREE = 生成された tree へのポインタ

処 理 概 要

buff は 2 バイト (FIXED BINARY (15, 0)) を単位とする配列である。



以下の処理で $\ell \leq 0$ となったとき

ppup[a, t, d] ; の t が求める tree のポインタとなる。

各 buffer の値によって次の様に処理する。

1) buff(j) = 1 ~ 2000 { operator node }

O P C = 1 ~ 2000 (operation code)
2 バイト

```

PPUP[a, rp, d];
PPUP[a, lp, d];
  AOPN.TYP ← OPN-NODE;
  {
    .LPTR ← lp;
    .RPTR ← rp;
    .OP ← buff(j);
  }
  t ← PUTNODE[AOPN-R];
  pdwn[o, t, NIL];
  j ← j + 1;      l ← l - 2;

```

} operator nodeの生成

2) buff(j) = 0 {NIL}

```

pdwn[0, NIL, NIL];
j ← j + 1;      l ← l - 2;

```

3) buff(j) = 2001~3977 {属性ノード}

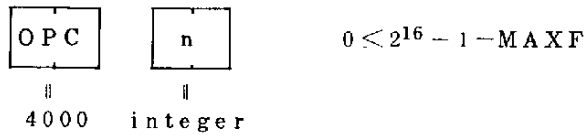
 .OPC = (lvar# * 100 + att-no) + 1900

```

d = buff(j) - 1900;        {101~2099}
lvar# = d / 100;
aH-no = d - lvar# * 100;
{属性ノードの生成}
  ATTN.TYP ← ATT-NODE;
  {
    .LPTR ← NIL;
    .RPTR ← lvar#;
    .OP ← att-no;
  }
  t ← PUTNODE[ATTN-R];
  pdwn[o, t, NIL];
  j ← j + 1;      l ← l - 2;

```

4) buff(j) = 4000 {埋め込み整数}

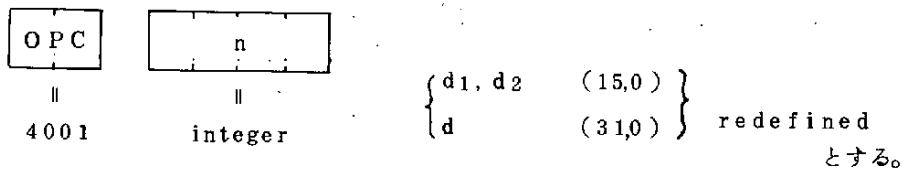


```

d ← buff(j+1) + MAXF;
pdwn[0, d, NIL];
j ← j + 2;      ℓ ← ℓ - 4;

```

5) buff(j) = 4001~4099 {整数定数ノード}

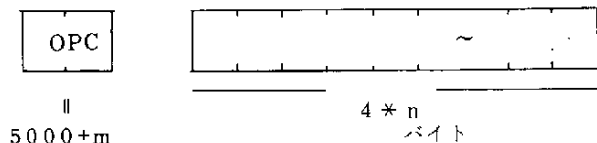


```

d1 ← buff(j+1);      d2 ← buff(j+2);
{定数ノードの生成}
CNTN.TYP ← CNT-NODE;
(
  .LPTR ← NIL;
  .RPTR ← buff(j)-4001; {ケタ数}
  .OP ← 2;
  DATA-D ← d;
t ← PUTNODE[CNTN-R];
pdwn[o, t, NIL];
j ← j + 3;      ℓ ← ℓ - 6;

```

6) buff(j) = 5001~5999 { 文字定数ノード }



{ d1, d2 (15.0) } redefined
 { dc CHAR(4) }
 とする。

```

cnt ← buff(j) - 5000;
j ← j + 1;      ℓ ← ℓ - 2;
tt ← NIL;
do while (cnt > 0);
{ 定数ノードの生成 }
  CNTN.TYP ← CNT-NODE;
  {
    .LPTR ← NIL;
    .RPTR ← if cnt ≥ 4 then 4;
              else cnt;
  }
  CNTN.OP ← 1;
  d1 ← buff(j);  d2 ← buff(j+1);
  CNTN.DATA-C ← dc;
  t ← PUTNODE[ CNTN-R ];
  if tt = NIL then pdwn[0, t, NIL];
    else LPTR[ tt ] ← t;
  tt ← t;
  cnt ← cnt - 4;  j ← j + 2;
                  ℓ ← ℓ - 4;
end;
{ cnt ≤ 0 で終了 }

```

7) buff(i) = 6000 ~ 6499

(PVノード)

OPC

= 6000 + n

{ PVノードの生成 }

```

ATTN.TYP ← PVN-NODE;
      .LPTR ← NIL;
      .RPTR ← NIL;
      OP ← buff(j) - 6000;
t ← PUTNODE[ATTN-R];
pdwn [0, t, NIL];
j ← j + 1;    l ← l - 2;

```

3.2.4 GCS定義例

GCS定義式から内部表現への変換の例を示す。GCS関係EMPLOYEEの定義式は次の様である。

```

DEFINE EMPLOYEE
(
  END, F-ENAME, L-ENAME, BIRTH-YEAR, BIRTH-PLACE,
  HIRE-YEAR )
(
  END=E1.END, F-ENAME=E1.ENAME, L-ENAME=E1.EFNAME,
  BIRTH-YEAR=E1.BIRTH-YEAR, BIRTH-PLACE=E1.BIRTH-PLACE,
  HIRE-YEAR=E1.HIRE-YEAR )
WHERE
  E1.END = E3.END AND E3.END = E4.END AND
  E1.END = E4.END ;

```

DIには、次の様な情報が格納される。

```

<STOREP> GREL IS STORED...EMPLOYEE          0600
OWNER IS EMPLOYEE      <STR_LVAR>
LVAR STORED...02 IN <STR_LVAR>
OWNER IS EMPLOYEE      <STR_LVAR>
LVAR STORED...03 IN <STR_LVAR>
OWNER IS EMPLOYEE      <STR_LVAR>
LVAR STORED...01 IN <STR_LVAR>
GATT STORED...01END      C0008END          0 IN <STR_LVAR>
GATT STORED...02F-ENAME  C0008F-ENAME      0 IN <STR_LVAR>
GATT STORED...03L-ENAME  C0008L-ENAME      0 IN <STR_LVAR>
GATT STORED...04BIRTH-YEAR C0008BIRTH-YEAR 0 IN <STR_LVAR>
GATT STORED...05BIRTH-PLACE C0008BIRTH-PLACE 0 IN <STR_LVAR>
GATT STORED...06HIRE-YEAR C0008HIRE-YEAR    0 IN <STR_LVAR>

```

GCS関係EMPLOYEEの定義式は、次の様な内部表現に変換される。

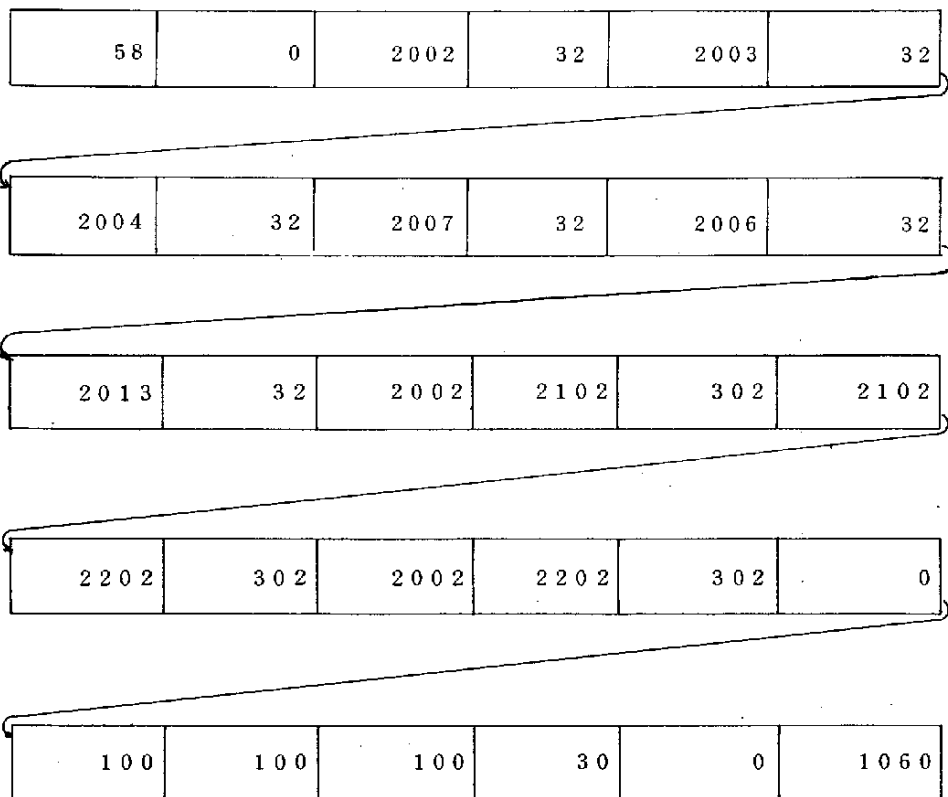


図3.9 GCS関係EMPLOYEEの内部表現

1～2000は、OPERATORノードを表わし、2001～3999は、属性ノードを表わす。又、0はNILノードを表わしている。

3.3 DQPの実現

3.3.1 DQPの概要

DQPは、GCS問合せ(GQ)の処理を行なう。利用者のGCS問合せを実行し、結果関係をRWSに格納し、利用者の要求に応じて結果を出力する。

DQPは、GCS問合せに対して次の処理を行なう。

- (1) 利用者のGQを入力して、表現変換して、全体LCS問合せ(グローバルLCS問合せ、GLQ)を得る。「表現変換」(ET)
 - (2) GLQから、各DM単位のLCS問合せ(LQ)を生成する。〔問合せ分割〕(QD)
 - (3) LQを各DMに放送する。
 - (4) 受け取ったLQをLDPに渡し実行させる。〔初期ローカル問合せ処理〕(ILQP)
 - (5) DM間結合処理の為に、GDP間での通信処理を行なう。〔DM間通信処理〕
- (1)~(4)は、結果DM(result DM、RDM)が行なう。(5)は、GQの処理に必要なデータを有しているDMの全てのGDPが行なう。

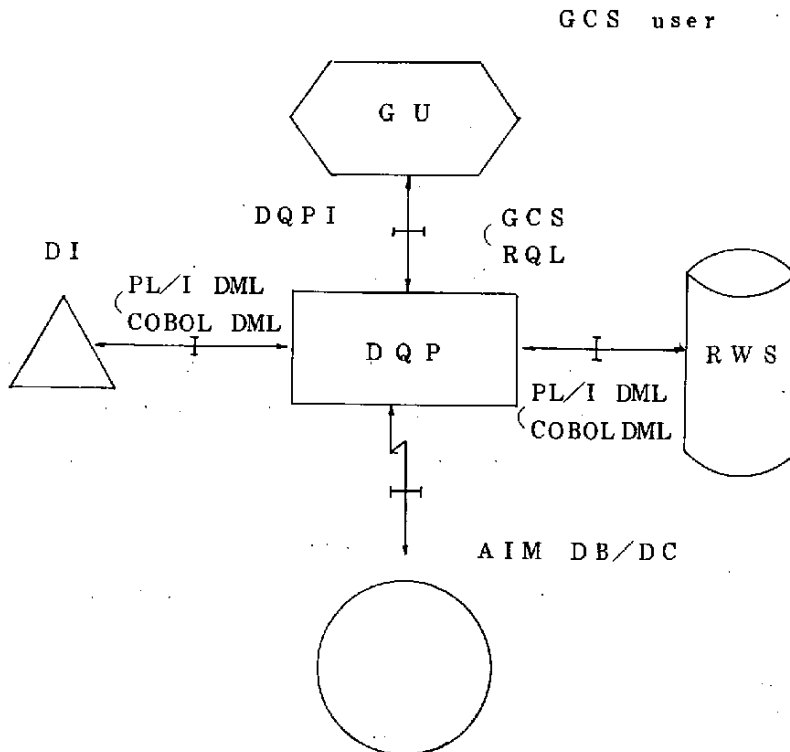


図 3.1 0 D Q P

3.3.2 DQPの実現

DQPの実現では、DQPをUSTとDMQPに分けて実現した。DQPの構成は、図3.11の様である。

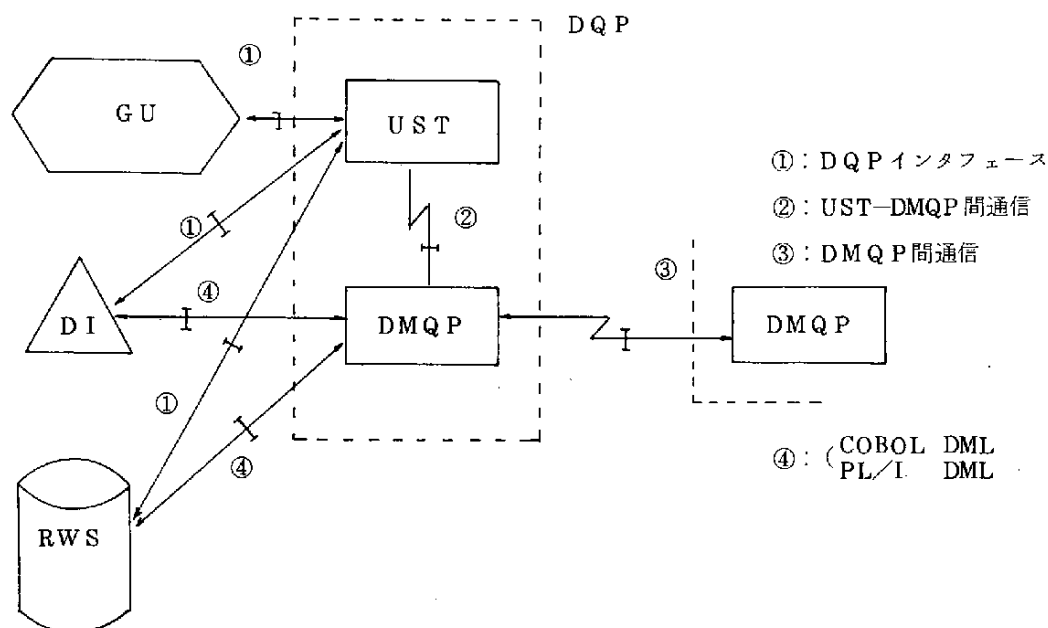


図 3.1 1 DQPの構成

USTは、GDPのDQPにおける表現変換、問合せ分割及び、ILQPのDMQPへの送信を行なう。

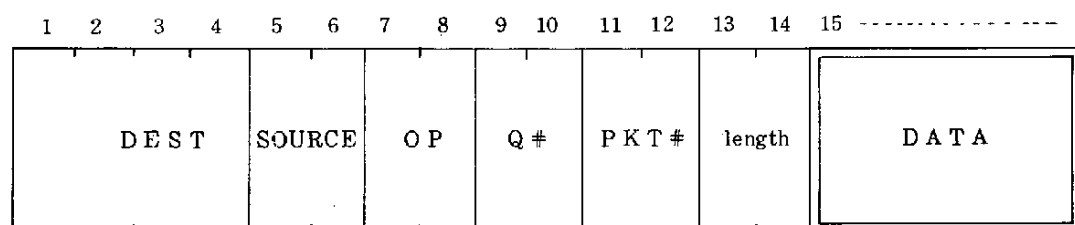
インタフェース①は、GCSのユーザに対するインタフェースである。DQPコマンドがこのインタフェースになる。DQPコマンドの詳細については、付記2 GDPコマンドを参照されたい。

インタフェース②は、USTとDMQPとのインタフェースである。これは、AIMのDB/DC機能を用いたモジュール間通信のメッセージで実現される。

DMQPは、DM間の通信処理と処理結果のUSTへの送信を行なう。

インタフェース③は、他のDMとのインタフェースである。AIMのDB/DC機能を用いたモジュール間通信のメッセージで実現される。

ここでは、インタフェース②と③に対して、そのコマンド（メッセージ）とその意味について述べる。インタフェース②、③ともにメッセージフォーマット（図3.12）は同一である。インタフェース②、又は③の区分はoperation codeによって分類される（表3.1）。



- DEST = 32 bits の bit-map で転送先の bit を ON にする。
- SOURCE = 送り手の DM #
- OP = メッセージのコマンド・コード
- Q # = 処理中の GQ #
- PKT # = packet 番号
- length = DATA 部のバイト長
- DATA = メッセージにより異なる。

図 3.12 メッセージフォーマット

表3.1 コマンド

コ マ ン ド	コマンド・コード	説 明
ILQP	1 0	DMQP間のILQP
ILQP-ACK	1 1	ILQPに対するACK通知
RELA	2 0	single attribute joinソース関係
REL-ACK	2 2	ソース関係に対するACK通知
REQ	3 0	ILQP処理要求
RACK	3 1	RECに対するACK通知
ABORT	1 3	ILQP異常終了通知
NOSOL	1 2	ILQPの結果なし通知
HAY	5 0	状態通知要求
IAM	5 1	状態通知
RSLT-ACK	4 0	結果に対するACK通知
NACK	3 2	REQに対する拒否通知
RELC	2 1	multi attribute joinソース関係

コ マ ン ド	コマンド・コード	説 明
ILQP	1 0 0	UST → DMQP ILQP処理要求
DONE	1 1 0	← ILQP終了通知
STATUS	2 0 0	→ 状態通知要求
ST	2 1 0	← 状態通知
ABORT	1 3 0	← 異常終了通知
NOSOL	1 2 0	← ILQP結果なし通知
KILL	3 0 0	→ 強制終了要求
KILLED	3 1 0	← 強制終了通知
BUSY	1 4 0	← ビジー通知
DNTKILL	3 2 0	← 強制終了拒否通知
ERR	3 9 9	← message sequence error
ID	9 9	→ user-id & gcs-name

3.3.3 メッセージフォーマット

USTとDMQP間、DMQPとDMQP間のインタフェースは、次の様なメッセージコマンドによって実現した。

(1) ILQPメッセージ

ILQPメッセージは、各DM単位のLCS問合せ(LQ)をコード化したメッセージ(図3.13)である。

GQを受けたUSTが、処理単位毎にILQPを生成し、DMQPへ送信する。

ILQPを受けたDMQPは、他のDMQPへREQメッセージを送信し、他のDMQPの確認を得る。全てのDMQPが受信可能ならば、他のDMQPにILQPを送信する。もし、DM内に対するLQがあれば、LDPを実行させる。

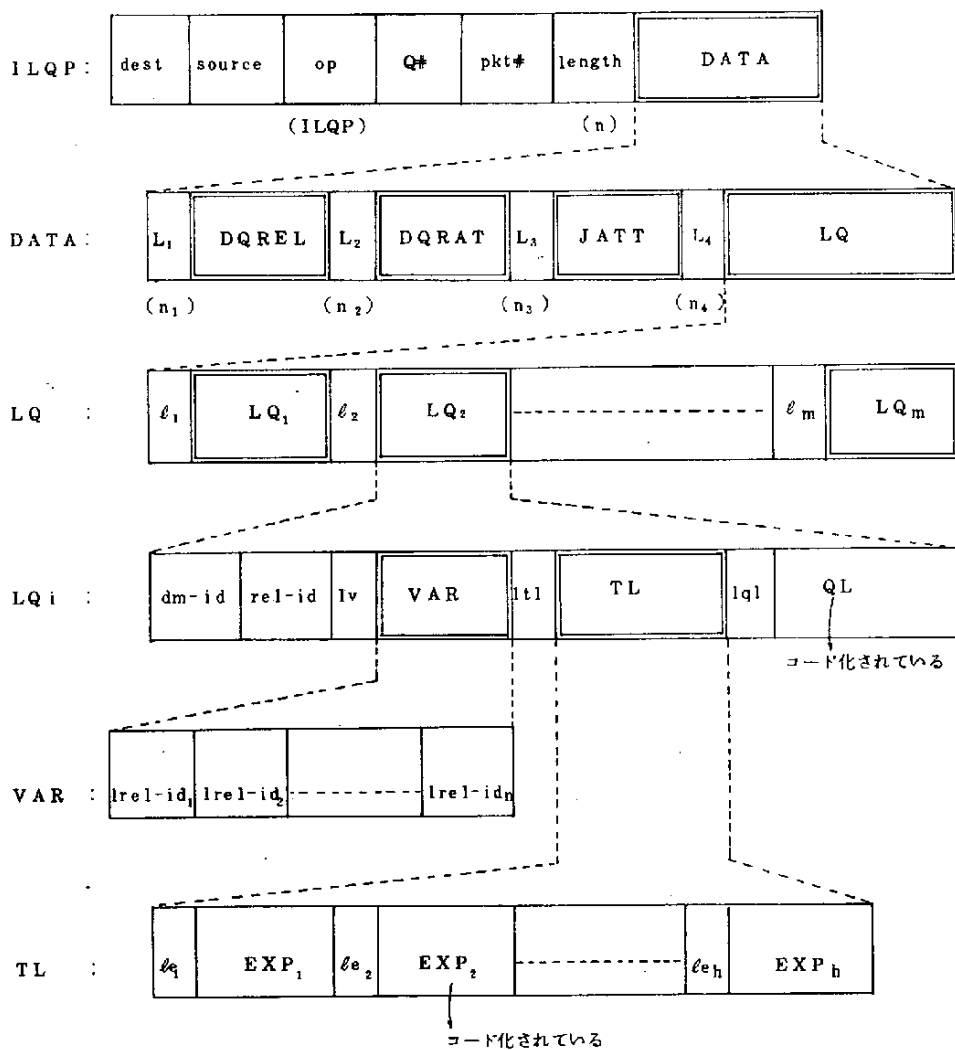


図 3.1 3 ILQPメッセージフォーマット(1)

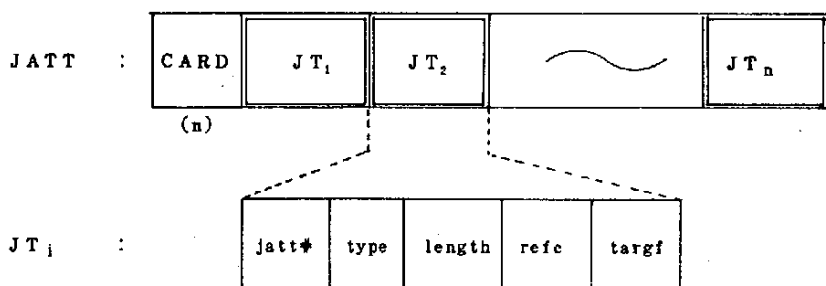
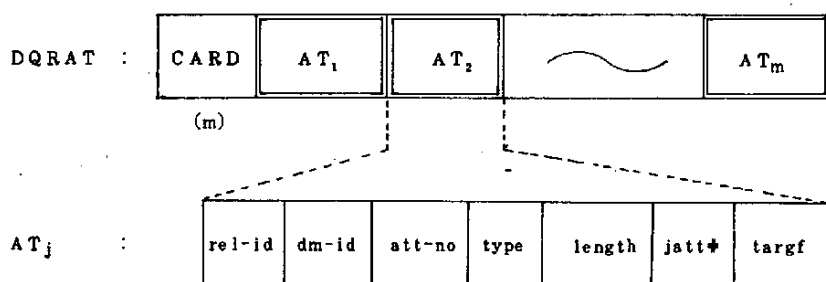
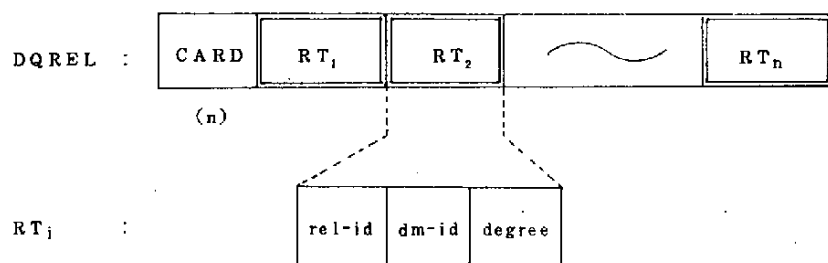


図3.14 ILQPメッセージフォーマット(2)

(2) ILQP-ACKメッセージ

ILQP-ACKメッセージ(図3.15)は、受信したILQPに対するローカル処理の結果を、他のDMQPへ通知するメッセージである。ローカル処理の終了は、RWSの結果関係の状態コードで判定する。

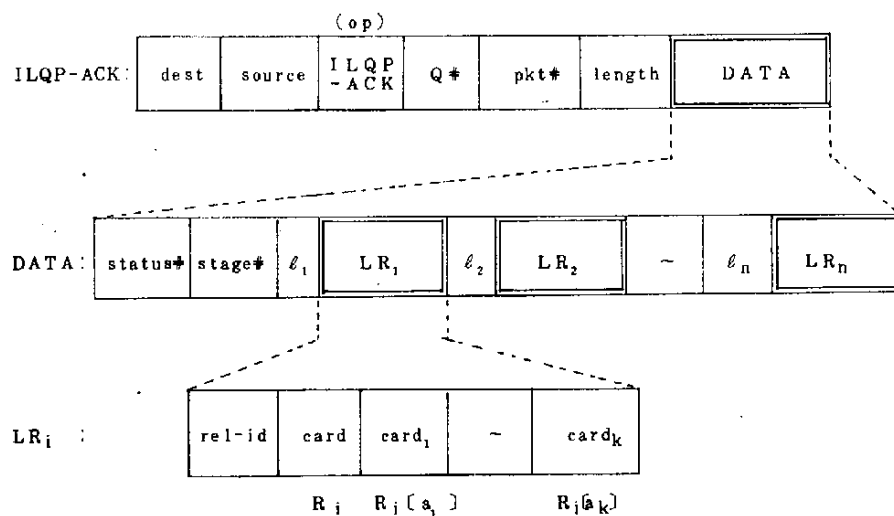


図3.15 ILQP-ACK

(3) NOSOLメッセージ

ILQPに対するローカル処理の結果が無かった場合に、他のDMQP、USTに通知するメッセージである(図3.16)。

このNOSOLメッセージを受けたUSTは、利用者に結果が無かったことを通知する。

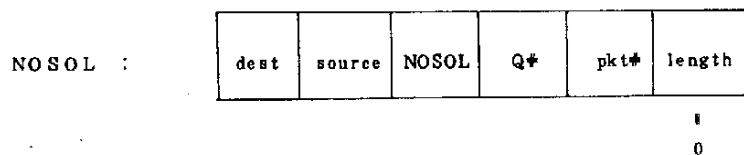


図3.16 NOSOLメッセージ

(4) RELメッセージ

ソース関係の転送用メッセージで、DM間結合処理の為に、他のDMQPへ送信されるメッセージである(図3.17)。RELメッセージには、単一属性(single attribute)のソースを転送するRELAメッセージと複数属性(multi-attribute)のソースを転送するRELCメッセージがある。

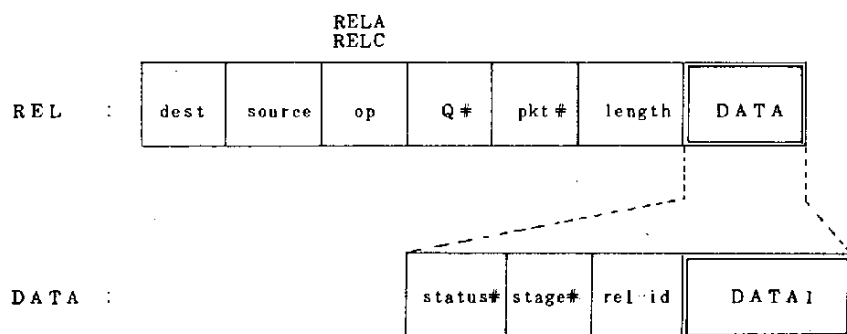


図3.17 RELメッセージ

どの関係がソースになるかは、DMQPが決定する。

j) RELAメッセージ

RELAメッセージは、基本的には図3.18の様である。データの圧縮によって次の3つの形式になる。

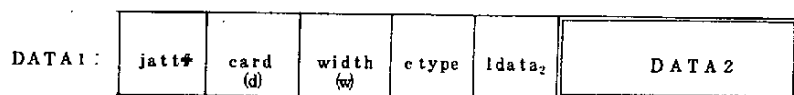


図3.18 RELAメッセージ

(i) ctype = 0 (normal)

属性の値をそのまま桁数分だけメッセージのデータ部分に埋め込む(図3.19)。

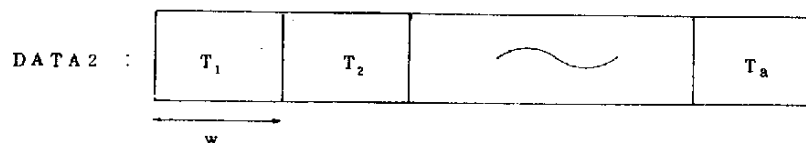


図3.19 RELAメッセージ(ctype=0)

(a) ctype=2 (右側のものをsuppress)

右側の空白を除いて、各値毎の桁長とともに値がセットされる(図3.2.0)。

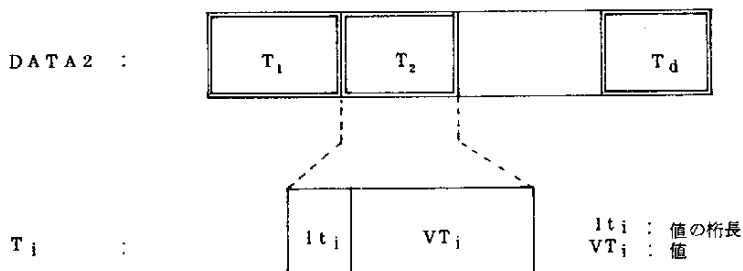


図3.2.0 RELAメッセージ(ctype=2)

(b) ctype=3 (front and rear compression)

ctype=2 と同じ形式で値がセットされる。

ii) RELCメッセージ

RELCメッセージの基本形式は図3.2.1の様である。ソース関係の組がコード化されているかによって次の2つの形式がある。

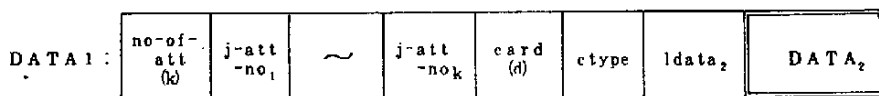


図3.2.1 RECLCメッセージ

(i) ctype=10

ソース関係の組がコード化されていない場合のRELCメッセージである(図3.2.2)

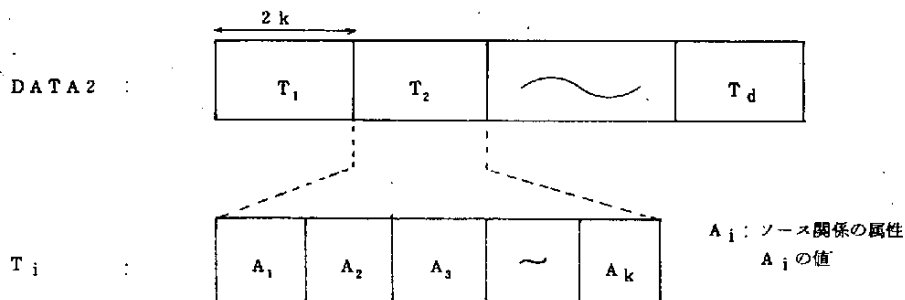
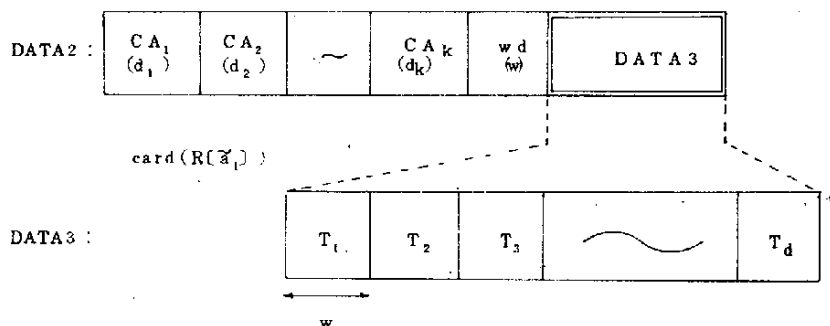


図3.2.2 RELCメッセージ(ctype=10)

(ii) ctype=11

ソース関係の組がコード化されている場合のRELCメッセージである(図3.2.3)



$$T_i = (\cdots ((v_1 \cdot d_2 + v_2) \cdot d_3 + v_3) \cdot d_4 \cdots + v_{k-1}) \cdot d_k + v_k)$$

図 3.2 3 RELCメッセージ (etype=11)

T_i は、コード化された組の値である。 d_i は属性 a_i のカーディナリティ、 v_i は属性 a_i の値である。

(5) REL-ACKメッセージ

RELメッセージに対して、DM間結合処理を実行した結果をビットマップで表わし、他のDMQPへ通知するメッセージである(図 3.2 4)

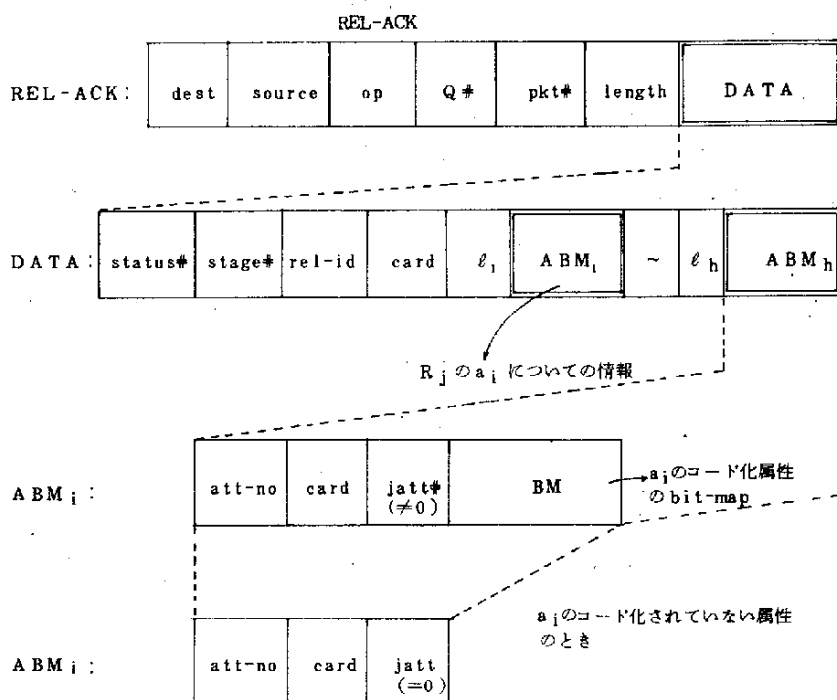


図 3.2 4 REL-ACKメッセージ

結合処理を実行した結果として、結合条件を満たす属性の値に対してはビットをオンにし、結合条件が満たされない属性の値に対してはビットをオフにする。

(6) REQメッセージ

USTからILQPメッセージを受けたDMQPが、他のDMQPにILQPの実行を要求する為のメッセージである(図3.25)。

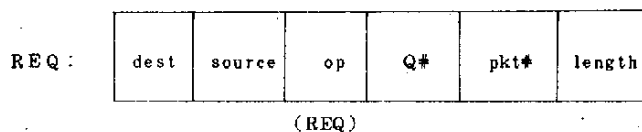


図3.25 REQメッセージ

(7) RACKメッセージ

REQメッセージを受けたDMQPが、ILQPの実行が可能な場合に、他のDMQPへ通知するメッセージである(図3.26)。

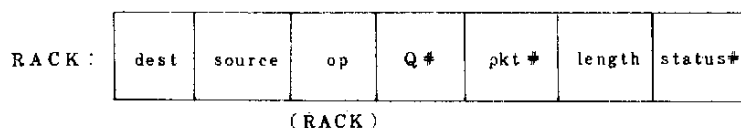


図3.26 RACKメッセージ

(8) ABORTメッセージ

ABORTメッセージは、ローカル問合せ(LQ)の処理が正常に終了しなかったことを検知したDMQPが他のDMQPへ通知するメッセージである(図3.27)。

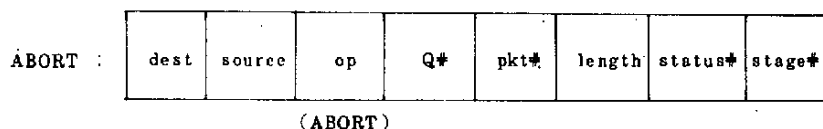


図3.27 ABORTメッセージ

(9) HAYメッセージ

ACKの返信の無いDMQPへ状態通知を要求するメッセージである(図3.28)。このメッセージを受けたDMQPは、IAMメッセージを送信する。

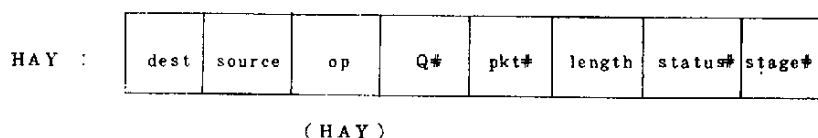


図 3.28 HAYメッセージ

(10) IAMメッセージ

HAYメッセージを受けたDMQPが現在の状態コードを他のDMQPに通知する為のメッセージである(図 3.29)。

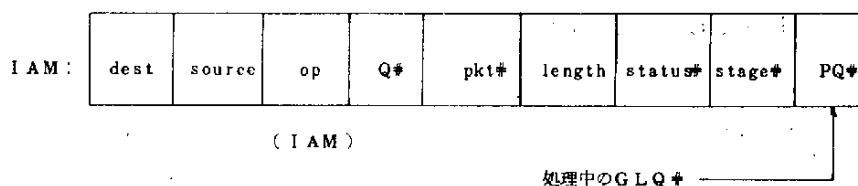


図 3.29 IAMメッセージ

(11) NACKメッセージ

REQメッセージに対する拒否メッセージである(図 3.30)。他のDMQPの1つでもNACKメッセージが返送された場合、REQ要求は失敗する。

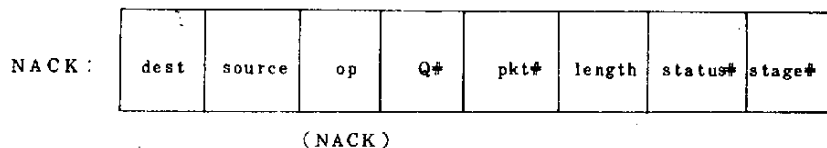


図 3.30 NACKメッセージ

(12) RSLT-ACKメッセージ

結果の転送に対するACKメッセージである(図 3.31)。

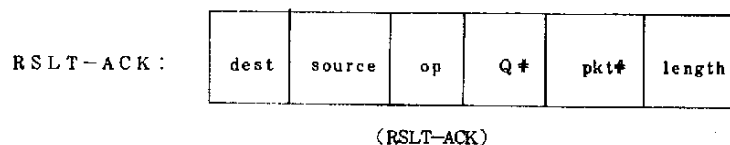


図 3.31 RSLT-ACKメッセージ

(13) DONEメッセージ

USTがDMQPに渡したILQPの処理が終了した事をUSTに通知するメッセージである(図3.3.2)。

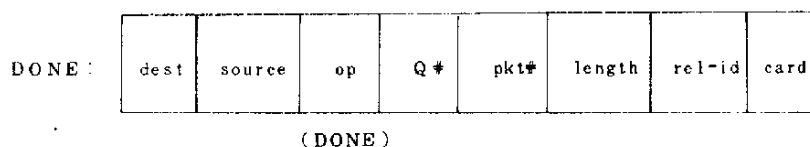


図 3.3.2 DONEメッセージ

(14) STATUSメッセージ

利用者からの状態通知要求をDMQPへ通知する為のメッセージである(図3.3.3)。

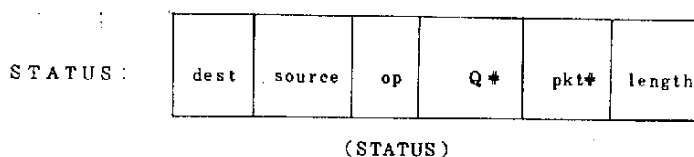


図 3.3.3 STATUSメッセージ

(15) STメッセージ

STATUSメッセージに対するDMQPからの返信メッセージ(図3.3.4)で、現在の状態をUSTに通知する。

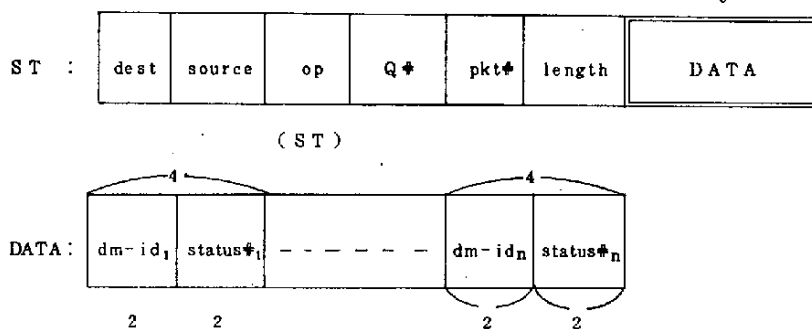


図 3.3.4 STメッセージ

(16) KILLメッセージ

利用者からの強制終了要求をDMQPへ通知するメッセージである〔図3.35〕。

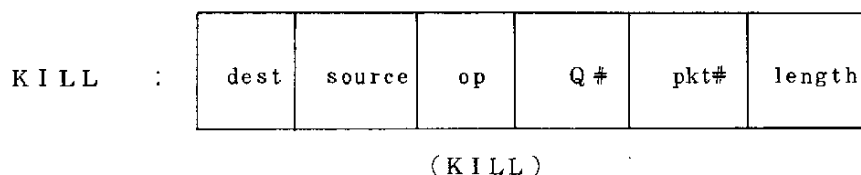


図 3.3 5 KILLメッセージ

(17) KILLEDメッセージ

DMQPからUSTへ強制終了を実行したことを通知するメッセージである〔図3.36〕。

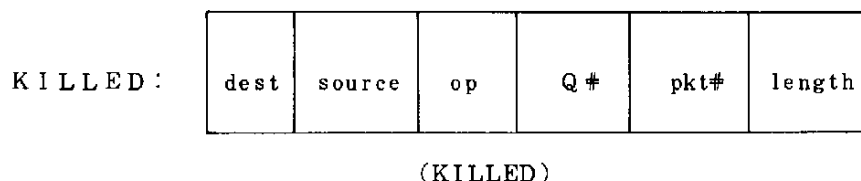


図 3.3 6 KILLEDメッセージ

(18) BUSYメッセージ

USTからのILQPメッセージに対して、すでに他のUSTのILQPを処理中で、要求が受けられないことを通知するメッセージである〔図3.37〕。

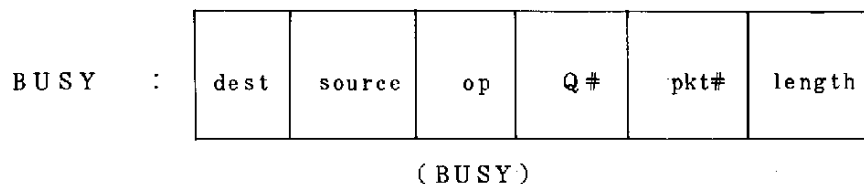


図 3.3 7 BUSYメッセージ

3.3.4 UST/DMQPの状態遷移

UST及びDMQPの通信処理における状態遷移について、その状態コードと処理概要を述べる。

A. USTの状態遷移

USTの状態遷移図は、図3.38の様である。各状態における処理の概要は以下の様である。

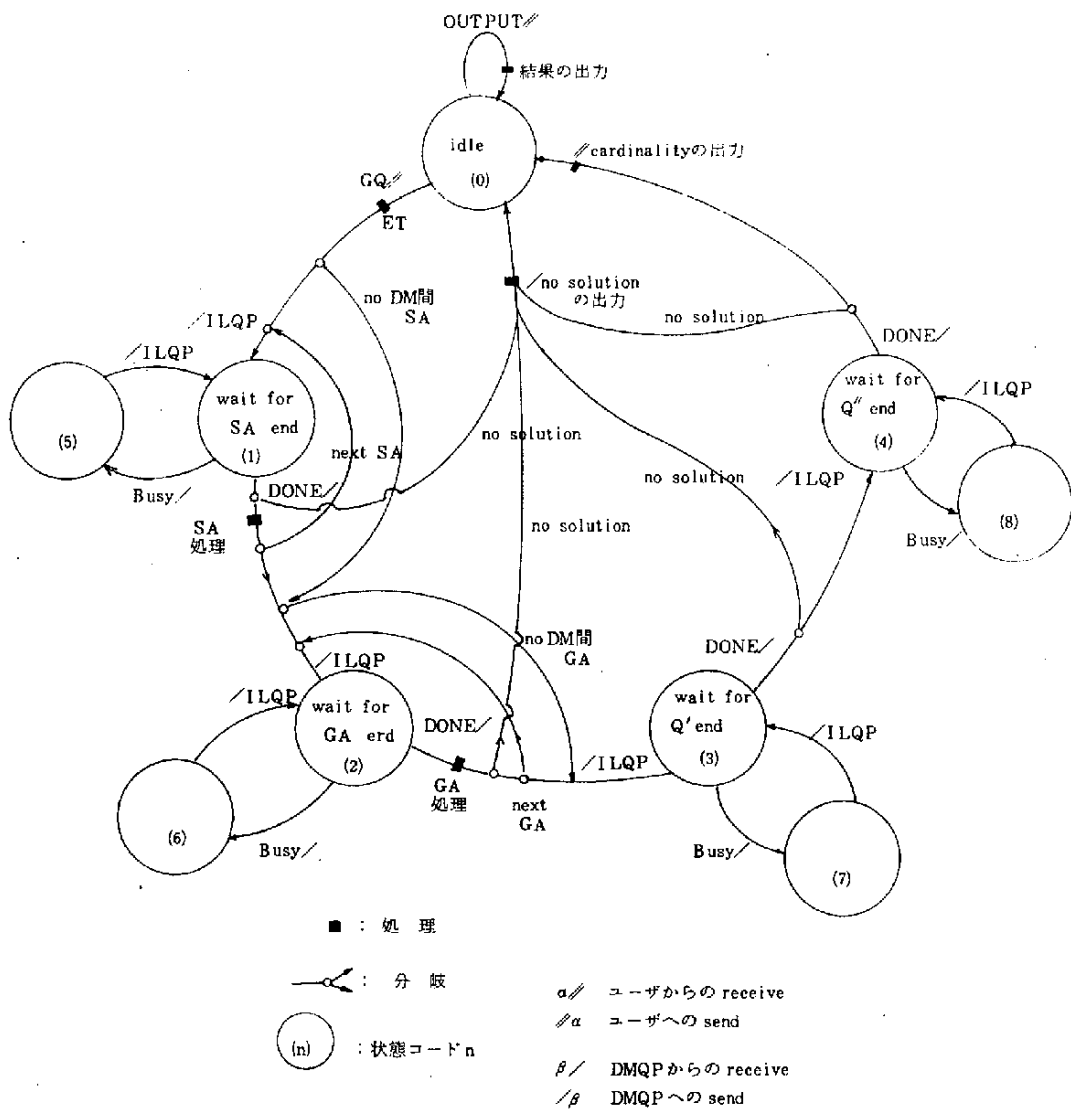


図 3.38 UST の状態遷移図

1) 状態コード=0 (メッセージ待ち状態)

利用者からのコマンド入力待つ。GCS 問合せの実行コマンドが入力されたら、表現変換を行ない、DMQPへILQPメッセージを送信する。ILQPメッセージを送信したら、状態コード1に遷移する。但し、DM間の simple aggregateが無い場合は、状態コード2に遷移する。

又、Output コマンドを受けた場合、RWS から結果関係をアクセスし、利用者に出力する。

2) 状態コード=1 (simple aggregate 処理終了待ち状態)

DMQPからのDONEメッセージを待つ。DONEを受信したら、simple aggregate 処理を行ない、次に simple aggregate があれば、再びILQPを送信して状態コード1に戻る。全ての simple aggregate を処理した場合、DM間のgroup-by aggregateに対するILQPを送信する。ILQPを送信したら、状態コード2に遷移する。

DM間のgroup-by aggregate がなければ、Q' に対するILQPを送信し、状態コード3に遷移する。

状態遷移0→1→2の処理概要を図3.39に示す。

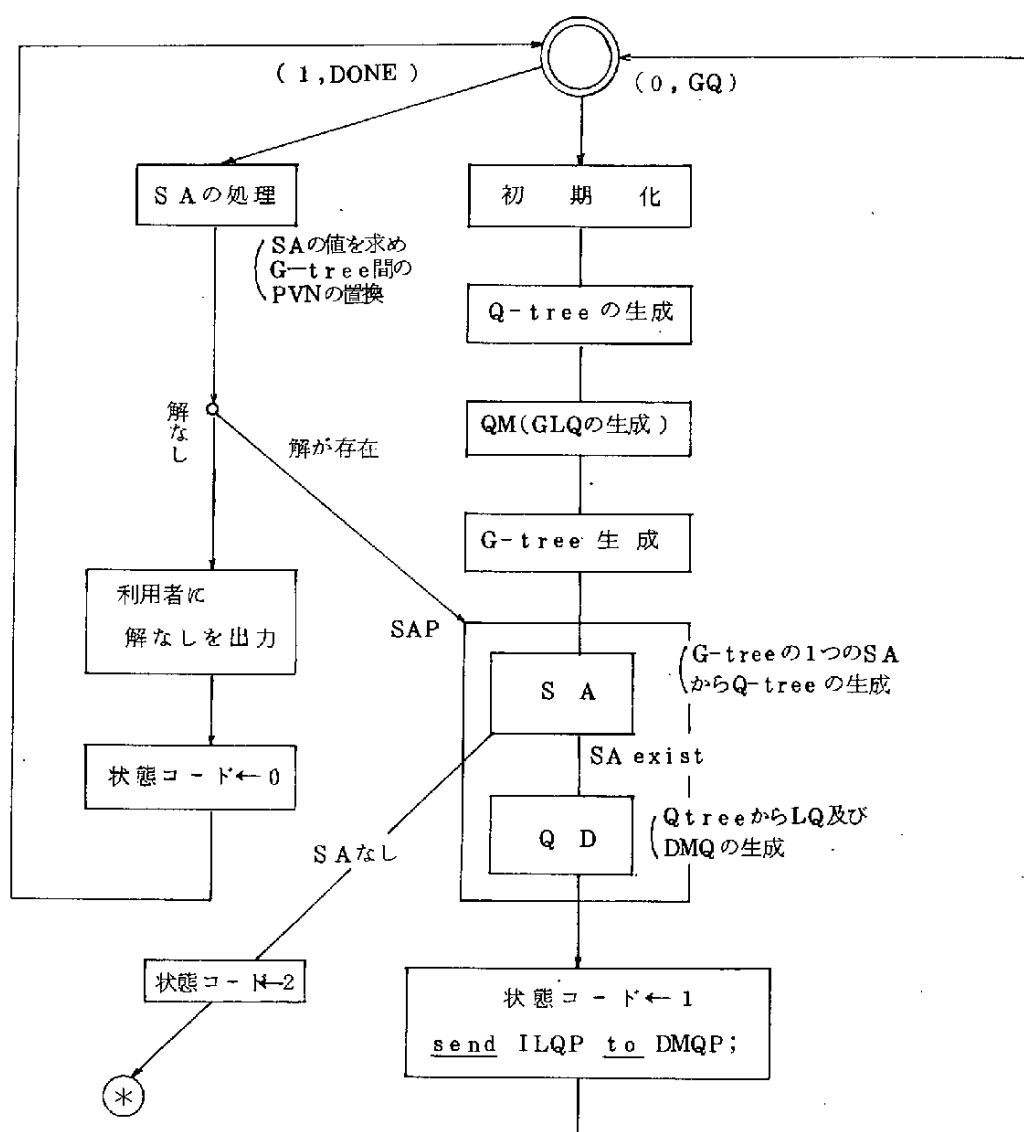


図3.39 UST状態遷移図 (0→1→2)

iii) 状態コード=2 (group-by aggregate処理終了待ち状態)

DMQPからのDONEメッセージを待つ。DONEを受けたら、group-by aggregate処理を行なう。他にgroup-aggregateがあれば、対応するILQPをDMQPへ送信し、状態コード2に戻る。

全てのgroup-by aggregateが済んだら、 Q^1 に対するILQPをDMQPに送信し、状態コード3に移移する。

状態遷移2→3の処理概要は、図3.40の様である。

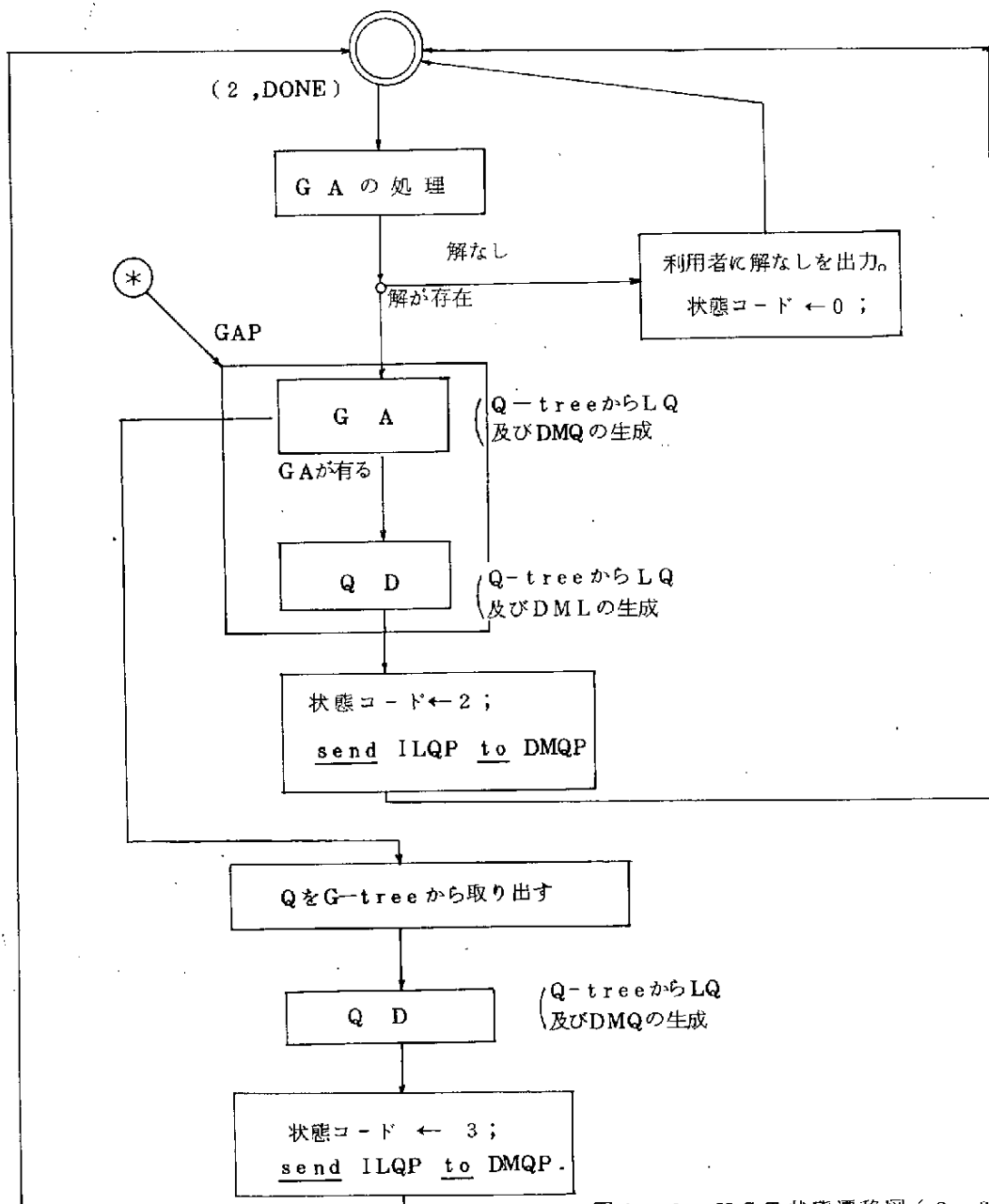


図 3.40 UST 状態遷移図 (2→3)

iv) 状態コード=3 (Q' 処理終了待ち状態)

DMQPからのDONEメッセージを待つ。DONEを受けたら、Q'' に対するILQPをDMQPに送信し、状態コード4に遷移する。

v) 状態コード=4 (Q'' 処理終了待ち状態)

DMQPからのDONEメッセージを待つ。DONEを受けたら、利用者にcardinalityを表示する。

状態遷移3→4→0の処理概要は、図3.4.1の様である。

状態コード1~4でNOSOLメッセージを受信した場合、利用者にno solutionメッセージを出力する。又、他のDMQPからBUSYメッセージを受けたら、各々状態コード5~8に遷移し、ILQPを再送し、元に戻る。

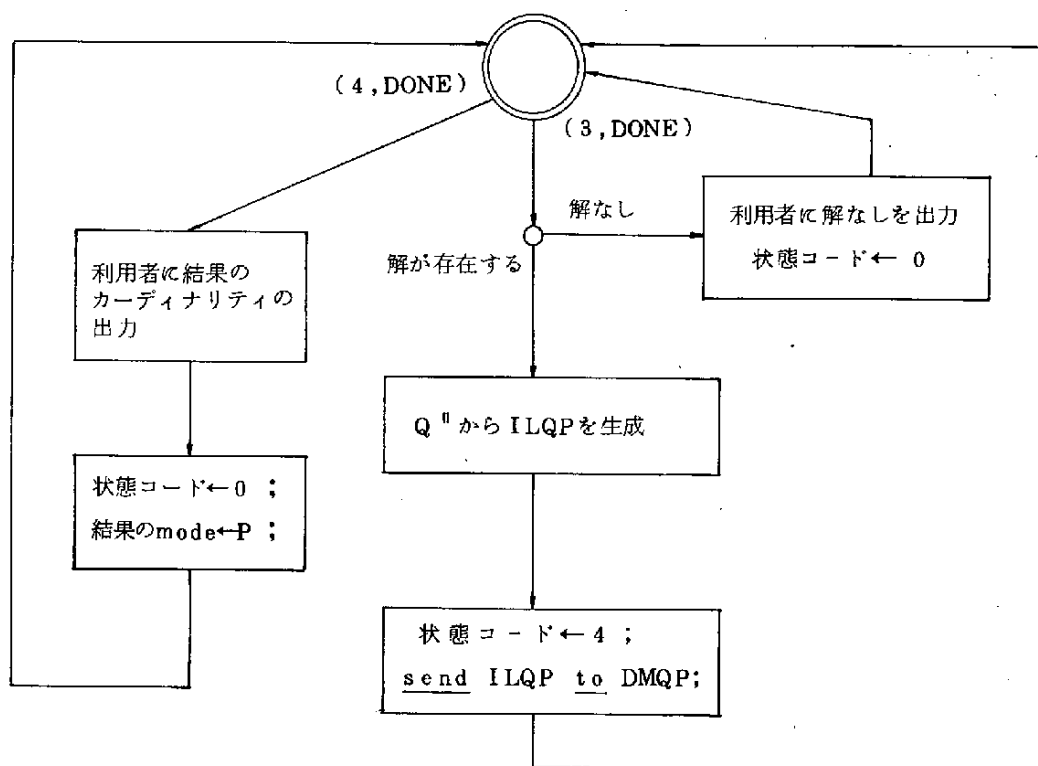
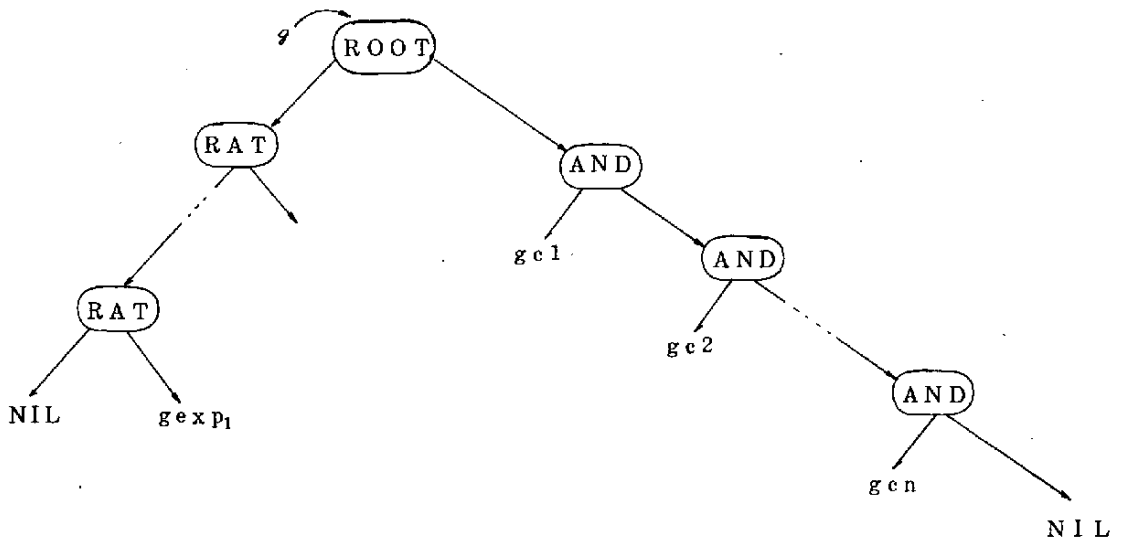


図 3.4.1 UST 状態遷移図 (3 → 4)

表現変換の処理概要

GCS問合せから、LCS関係を参照する全体LCS問合せ(GLQ)に表現変換(ET: Expression Transformation)を行う。状態遷移図3.38のETの処理概要を以下に示す。



1) GCS問合せ木(g)にvariableのGRTBLに対するbit-mapをsetする。

Call CONCTCK[g];(現在のCONCTCK)

2) gが参照している全てのGCS variableのbit-mapを求める(bmとする)。

$bm \leftarrow BVL[g] \cup BVR[g];$

3) GRRTBLのmarkをoffする。

4) bit-map(bm)内でbitがonであるbit(variable)について($\forall i(bm(i) = ON)$ について)以下の処理を行う。

4-1) $rn \leftarrow GRTBL_rel_name[i];$

4-2) $gr = 0$ (GRRTBLにない)とき

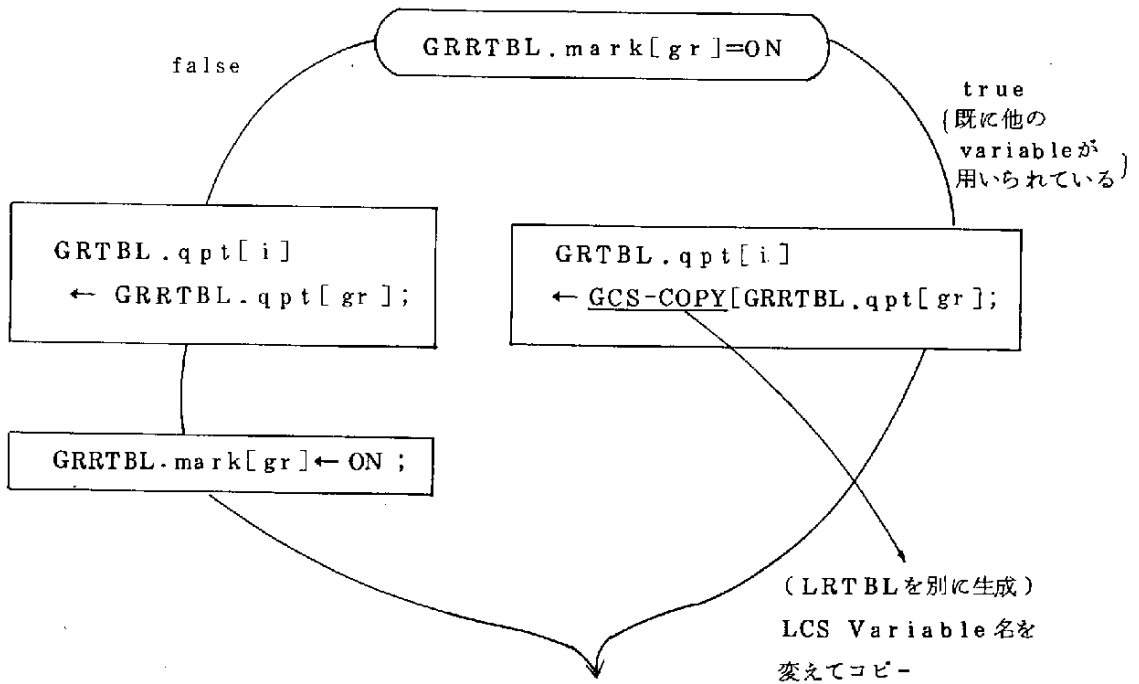
DIからGCS定義式をloadする。

$P \leftarrow load - GCS[rn];$

{ GRRTBL, GRATBL, LRTBLにset }

$gr \leftarrow CRRTBL; mark[gr] \leftarrow off;$

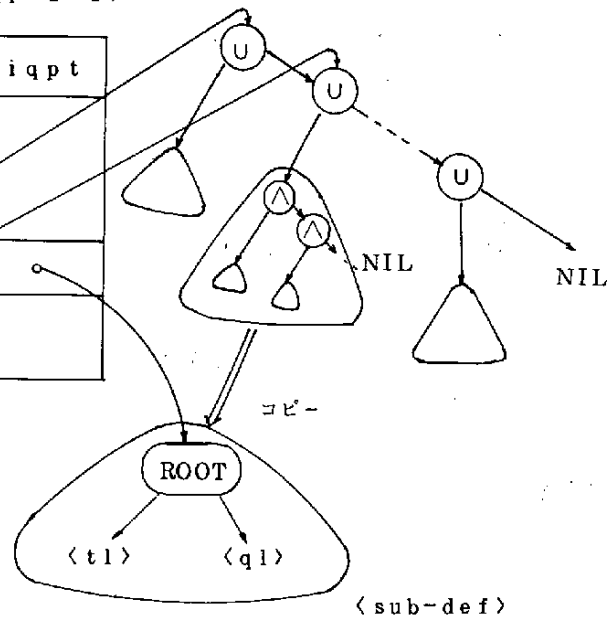
4-3) $gr > 0$ のとき



5) GRTBL qptをuqptにstoreする。

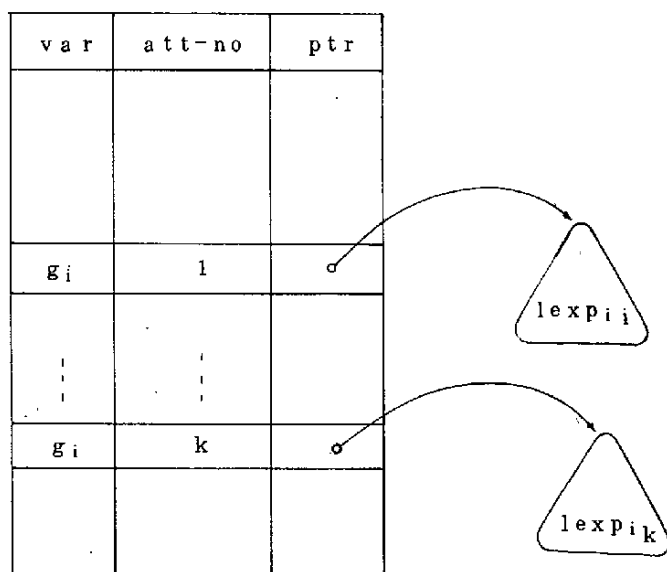
GRTBL GRTBL[i] ← GRTBL.qpt[i];

var		qpt	uqpt	iqpt
"gi"				

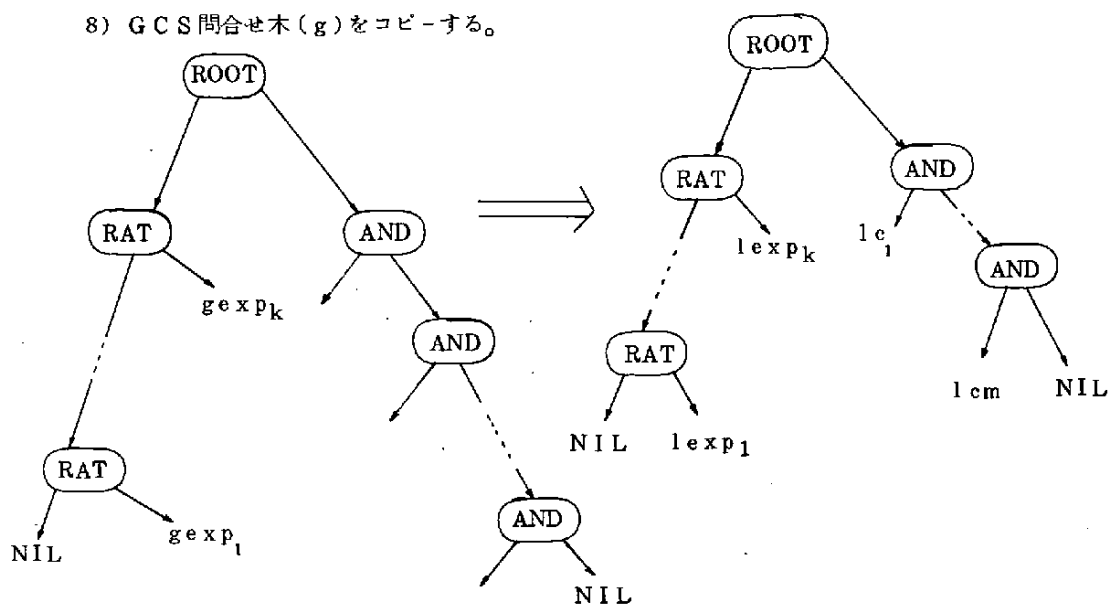


- 6) $G_i, \dots, G_m$ を、GCS 問合せ(q)の参照しているGCS定義式の〈sub-def〉の組み合わせとする。〈sub-def〉をコピーしてGRTBL, $i q p t$ にポインタを set する。このとき LATT table を用いて属性ノードを管理する。以下ある G_i についての処理を示す。
- 7) 各〈sub-def〉をコピーするとき、target-listをサーチして $a_{ij} = \text{exp}_{ij}$ に対して g_i, a_{ij} と exp_{ij} に対応をATTBLに生成する。

ATTBL



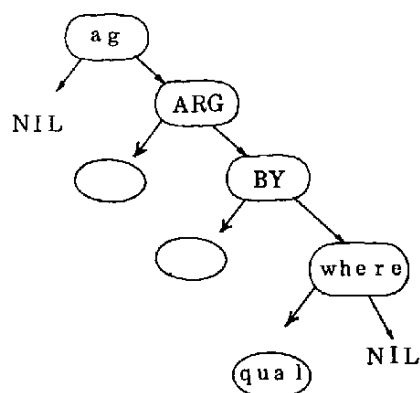
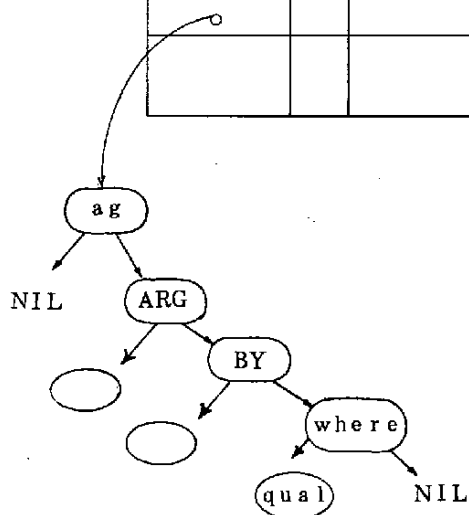
- 8) GCS 問合せ木(g)をコピーする。



- 属性ノードについてはATTBLをサーチし、(lexp)でreplaceする。
- aggregate node (OPN - NODEでOP=1001~1029)については、ノードをコピーするとともに、AGRTBLにsetする。

AGRTBL

gptr		lptr



9) fvar(g) の変数の qual(g) を conjunct する。

bmg ← fvar [g];

$\forall_R (bmg(k) = ON)$ について

GRTBL, iqpt[k] の qual と RPTR(nq) を接合する。

10) AGRTBL の各 tuple について (j とする)

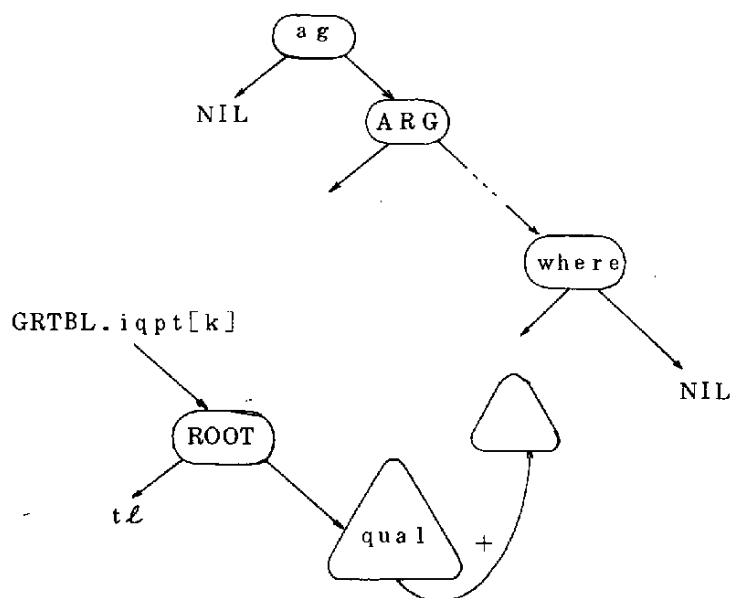
bmg ← fvar [AGRTBL.gqpt[j]];

$\forall_k (bmg(k) = ON)$ について

GRTBL. iqpt [k] の qual と AGRTBL. lqpt[j]

内の qual を接合する。

AGRTBL. ℓ_{qpt} [j]



11) COPY(modify) された tree のポインタ (ng) を GRTBL. ℓ_{qrt} に set する。

B. DMQPの状態遷移

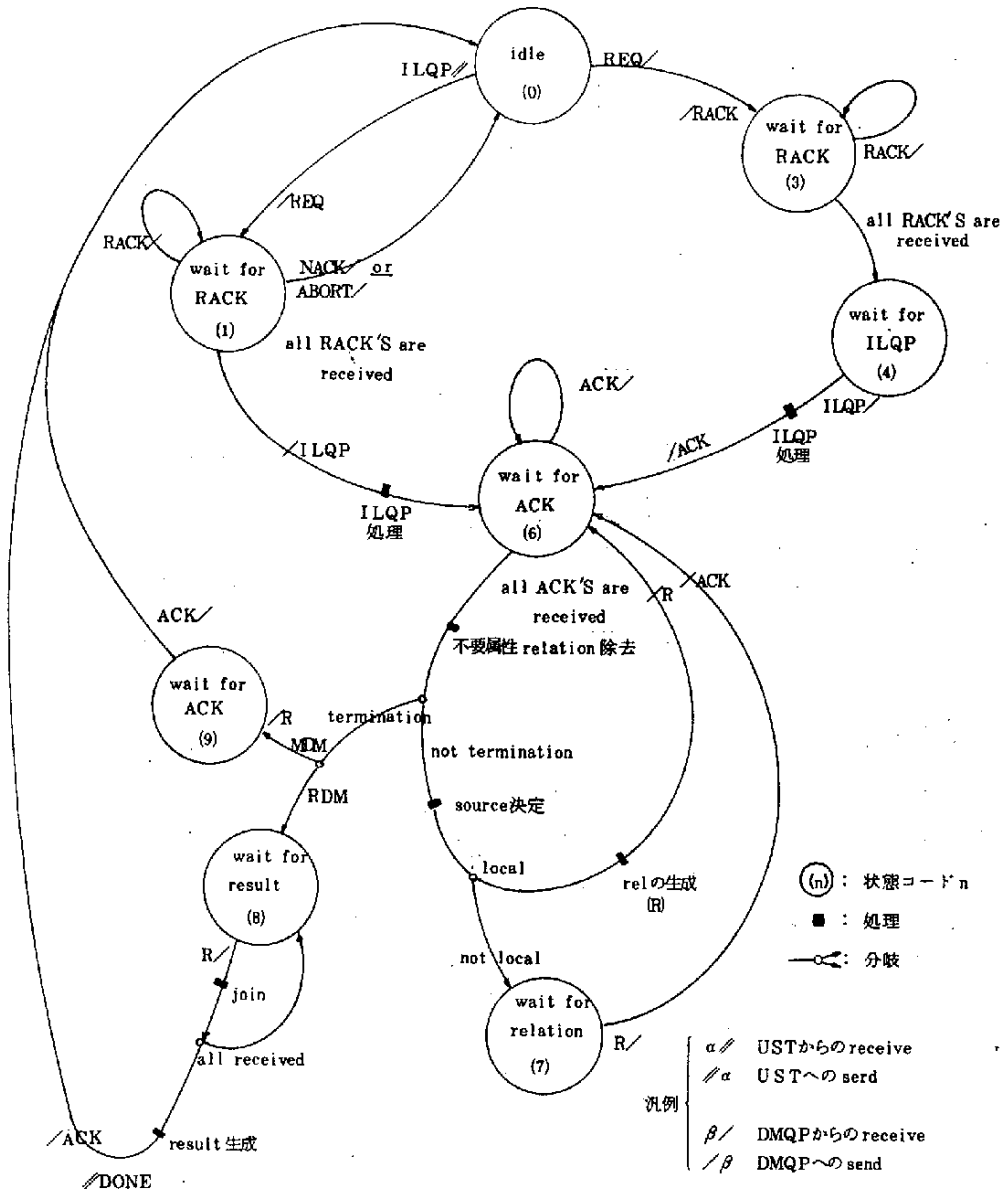


図 3.4 2 DMQP 状態遷移図

図 3.4 2 DMQP 状態遷移図

DMQPの状態遷移図は図3.42の様である。各状態における処理の概要は、以下の様である。

i) 状態コード=0 (idle状態)

DMQPは、起動されるとメッセージファイルのread状態で、メッセージの受信を待つ。受信したメッセージがUSTからのILQPであるのか、他のDMQPからのREQであるかによって状態が分岐する。

ILQPの場合、他のDMQPにREQメッセージを送信し、状態コード1に遷移する。この時の処理概要は、図3.43の様である。

REQの場合、他のDMQPにRACKメッセージを送信し、状態コード3に遷移する。この時の処理概要は、図3.44の様である。

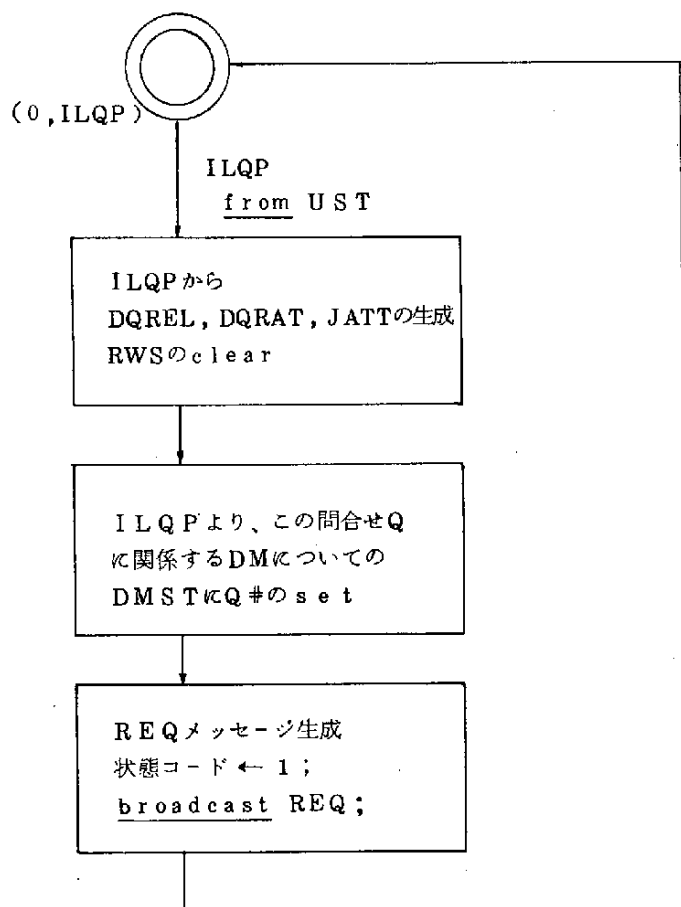


図 3.43 DMQP状態遷移図(0→1)

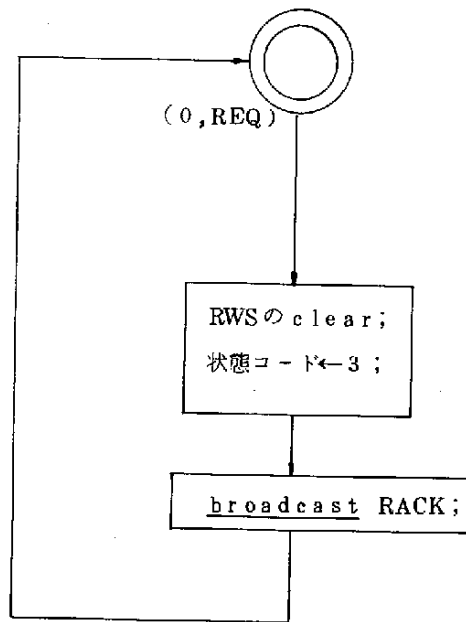


図 3.4 4 DMQP 状態遷移図 (0→3)

ii) 状態コード=1 (RACKメッセージ待ち状態)

他のDMQPからのRACKメッセージを待ち、全てのDMQPからRACKを受信したら、ILQPを他のDMQPに送信する。自dm内にローカル問合せがあれば、ILQP処理(ローカル問合せの生成とLDPの実行)を行う。ILQP処理が終了したら、ACKを生成し、他のDMQPに送信する。ACKを送信したら、状態コード6に遷移する。

但し、NACK又はABORTメッセージを受信した場合は、状態コード0に戻る。

この時の処理概要は、図3.45の様である。

iii) 状態コード=3 (RACKメッセージ待ち状態)

他のDMQPからのRACKメッセージを待ち、全てのDMQPからRACKメッセージを受けたら状態コード4に遷移する。

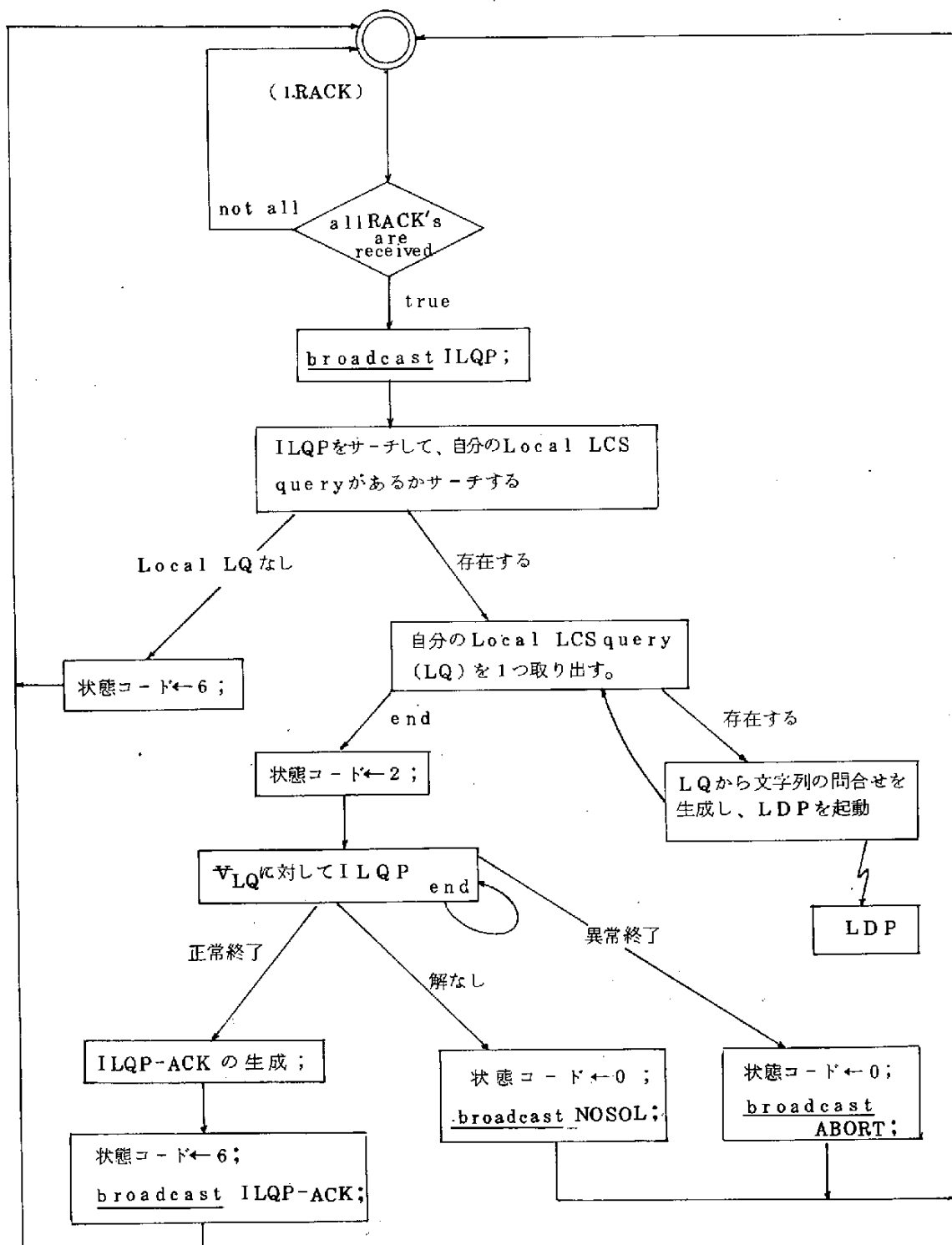


図 3.45 DMQP 状態遷移図 (1→6、1→2→6、1→0)

Ⅳ) 状態コード=4 (ILQPメッセージ待ち状態)

REQメッセージを送信したDMQPからILQPメッセージが送信されるのを待つ。ILQPメッセージを受けたら、自DM内のローカル問合せがあれば、ILQP処理を実行する。ローカル問合せ処理が終了したら、ACKを生成し他のDMQPに送信し、状態コード6に遷移する。

この時の処理概要は、図3.46の様である。

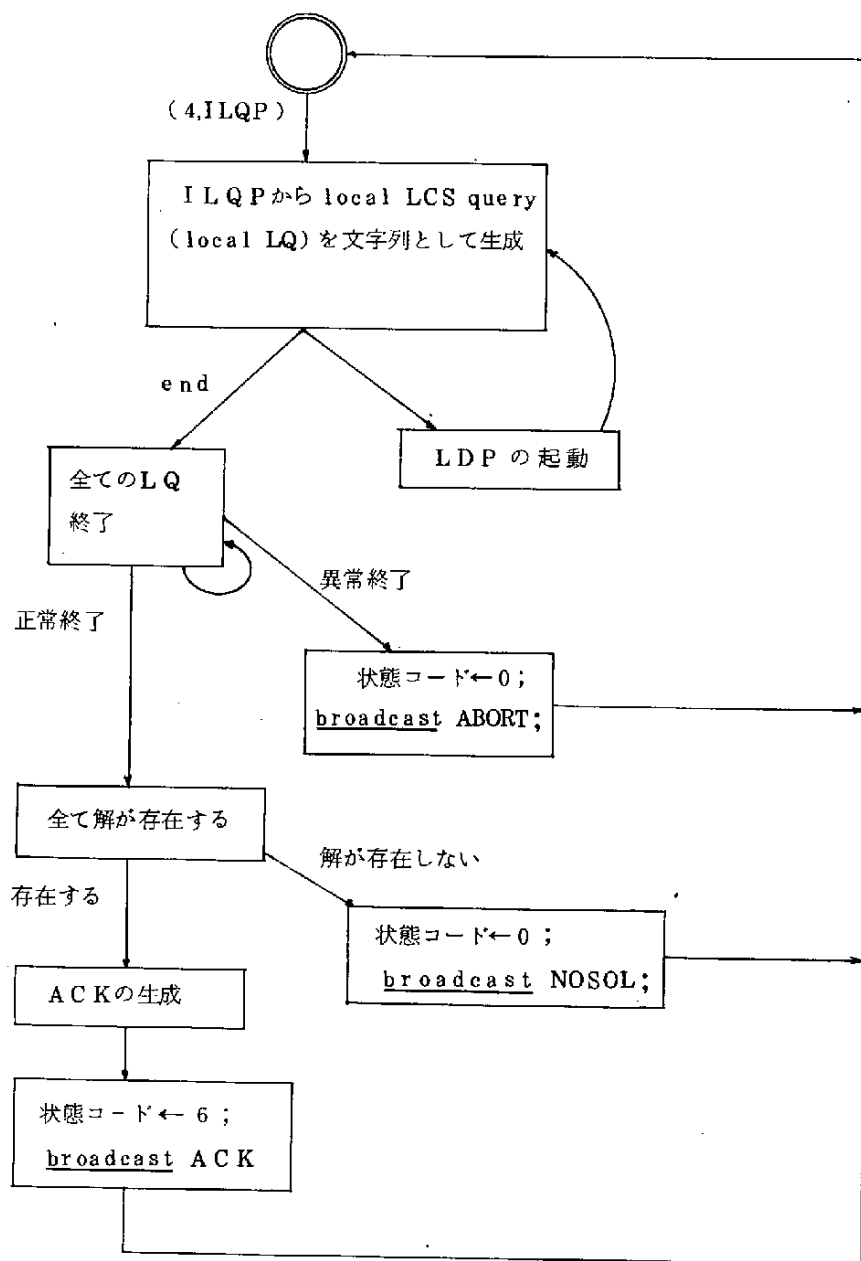


図 3.46 DMQP 状態遷移図 (4→6)

V) 状態コード＝6 (ACKメッセージ待ち状態)

他のDMQPからのACKメッセージの受信を待ち、全てのACKを受けたら、不要属性の除去を行う。次に、GCS問合せ処理の終了判定を行う。もし終了でなければ、次のソース関係を決定する。ソース関係が自サイト内であれば、RELメッセージを生成し、他のDMQPへ送信する。RELメッセージを送信したら、状態コード5に遷移する。

自サイト内にソース関係がなければ、状態コード7に遷移する。

もし、処理が終了している場合、自DMかRDMであれば、状態コード8へ、MDMであれば結果をRDMへ送伝し、状態コード9へ遷移する。

この時の処理概要は、図3.47の様である。

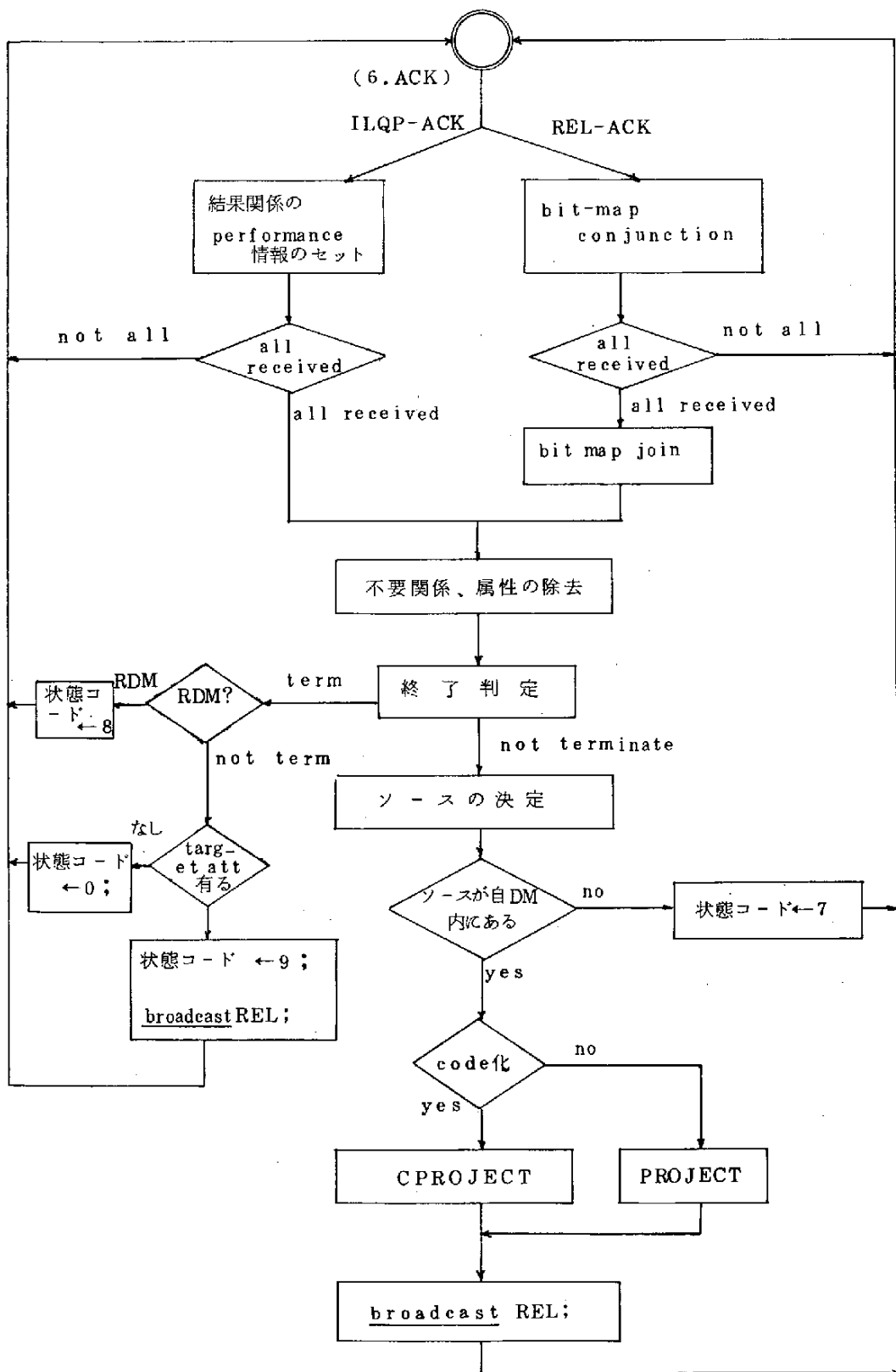


図 3.47 DMQP 状態遷移図 (6→7、6→9、6→8)

VD) 状態コード=7 (RELメッセージ待ち状態)

RELメッセージを待ち、受信したら結合を行なう。結合結果からACKを生成し、他のDMQPへ送信し、状態コード6へ遷移する。

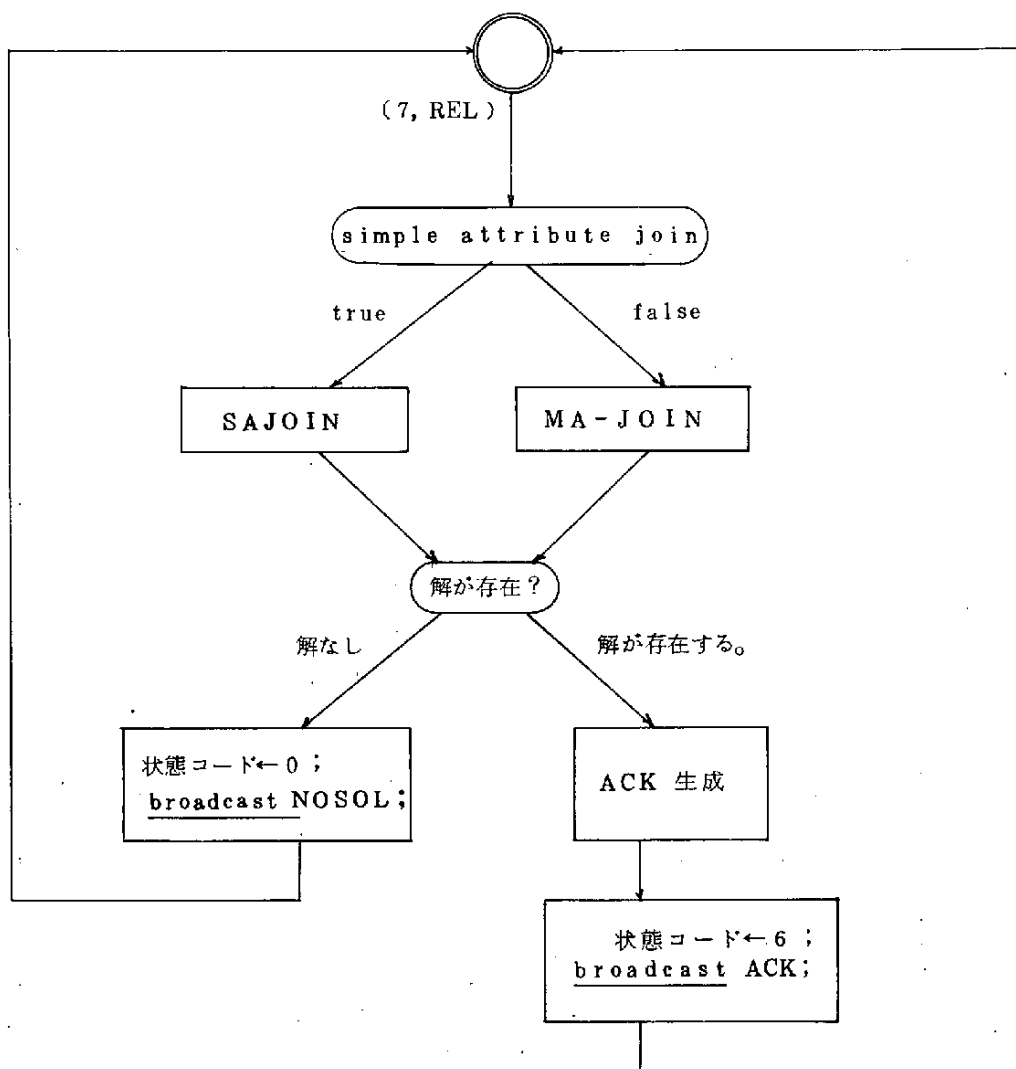


図 3.4 8 DMQP 状態遷移図 (7→6)

vii) 状態コード=8 (結果待ち状態)

他のDMQPからの結果を待ち、結果を受信したら、結合処理を行う。全ての結果を受けたら、USTにDONEメッセージを送信する。又、他のDMQPにACKを送信し、状態コード0に戻る。

この時の処理概要は、図3.49の様である。

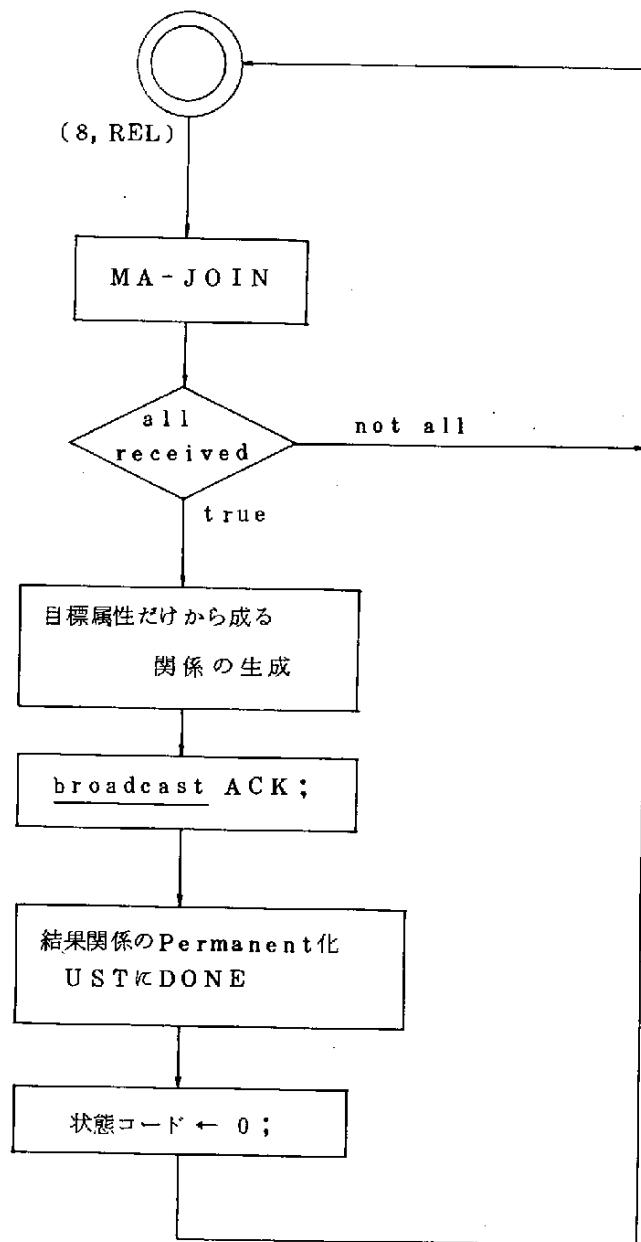


図3.49 DMQP状態遷移図(8→0)

viii) 状態コード=9 (ACKメッセージ待ち状態)

ACKメッセージを待ち、受信したら状態コード0に戻る。

3.3.5 モジュール間通信の実現

GDPのアーキテクチャは、図3.50の様である。USTは、TSS配下のジョブとして実行される。利用者とUST間には、DQPインタフェースが提供される。

モジュール間通信としては、USTとDMQP、DMQPとDMQP、DMQPとLDP間の通信がある。モジュール間通信は、AIMのDB/DC及びインタナルリダを用いて実現した。

i) AIMのDB/DC

USTとDMQP、DMQPとDMQP間の通信をAIMのDB/DCを使って実現した。GCS問合せを実行する場合、DMQPは予めバッチジョブとして起動しておく事が必要である。DMQPは、UST又は他のDMQPからのメッセージを待つ。メッセージを受けて処理が実行される。

ii) インタナルリダ

DMQPから、ローカル問合せ(LQ)を実行するためのLDPの起動及びLQの受け渡しをインタナルリダによって実現した。又、LDPがLQから生成したCOBOL DMLプログラムの起動もインタナルリダによって実現した。LDPは、必要な時だけ(ローカル問合せがある時だけ)、DMQPより起動される。

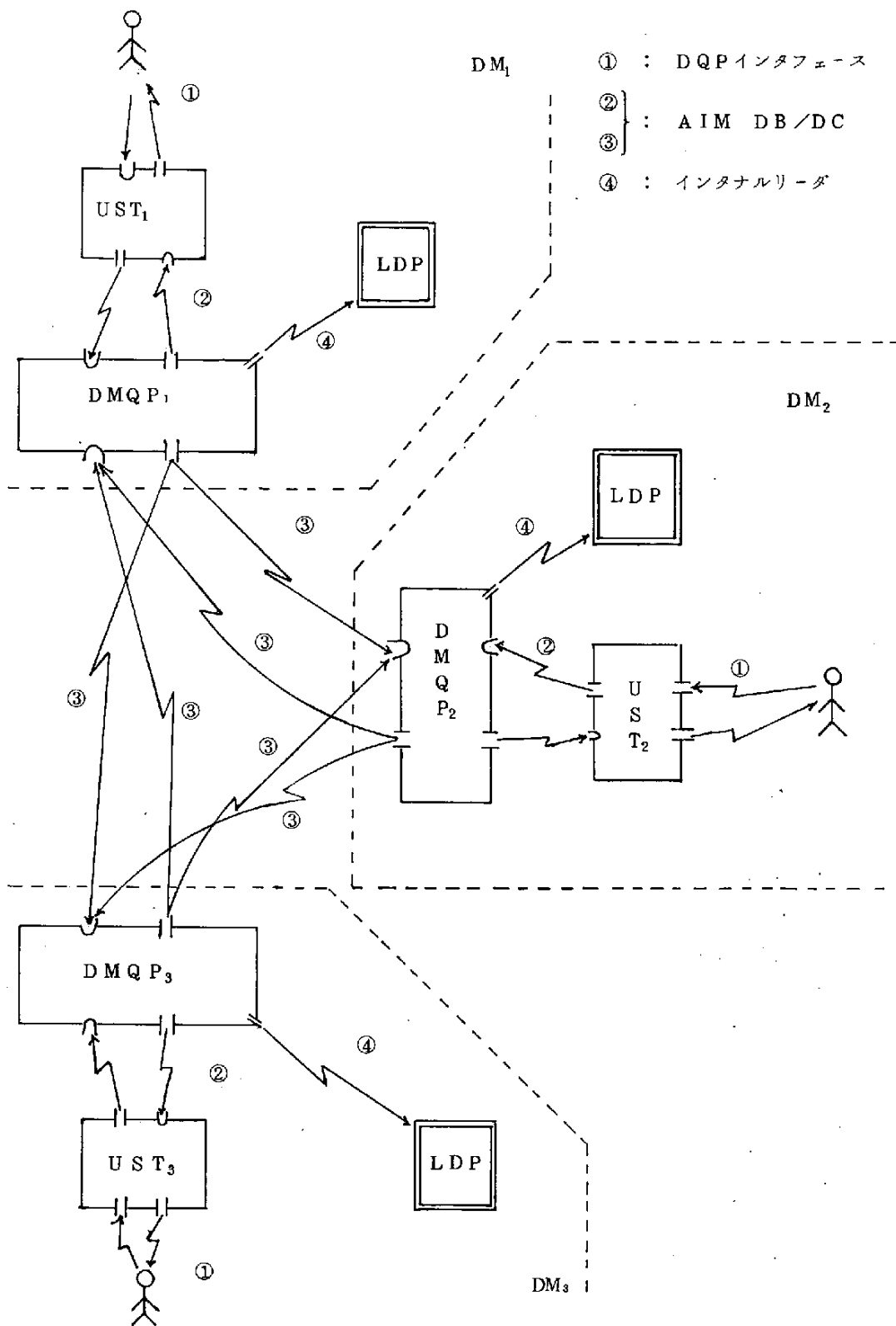


図 3.50 GDP のアーキテクチャ

A. AIMのDB/DC機能

AIM配下の応用プログラム間の提供される通信機能（DC機能）を用いて、USTとDMQP、DMQPとDMQP間の通信を実現した。DC機能を使った通信処理ではモジュール間の1対1通信が可能である。モジュール間の通信は、メッセージファイルへのRead/Writeによって実行される。通信相手のモジュールを識別する為に論理宛先が使用される。論理宛先は、メッセージの宛先名である。論理宛先は、メッセージファイルと対応づけられる。論理宛先に対して送出されたメッセージは、対応づけられたメッセージファイルをReadするモジュールに読み込まれる。

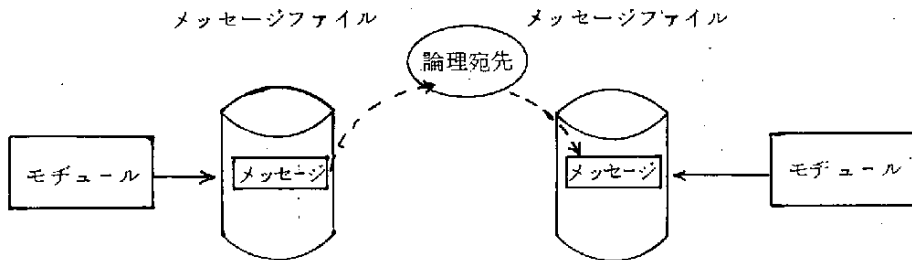


図 3.5.1 モジュール間の通信

AIM DB/DCを使ったUST、DMQP間の通信処理の概要を、図 3.5.2 に示す。

これらを実現する為には、次の様な環境を設定する。

(1) DCMS定義

DCMS（DC管理システム）全体に関する情報（システムの規模、DCMSバッファの設定等）を定義する。

(2) PROCEDURE定義

業務（プロシジャ）や応用プログラムに関する情報（メッセージ キューノードの設定等）を行なう。

(3) LD定義

論理宛先（LD）に関する情報を定義する。

(4) PED定義

応用プログラムの動作環境を定義する。

AIMでの環境設定の例を、以下に示す。

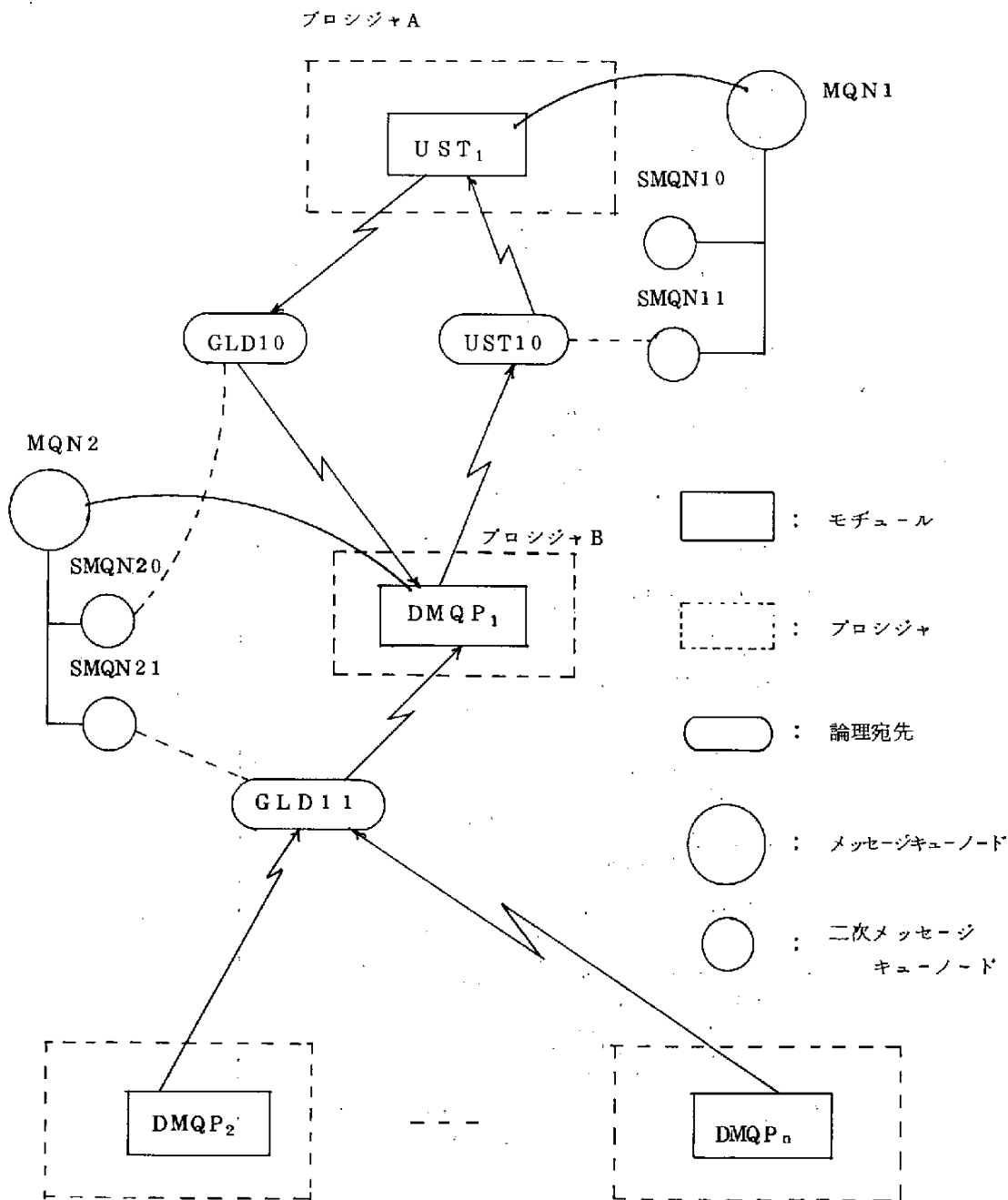


図3.52 AIM DB/DCによる通信処理の概要

1) DCMS定義

```

*      ****
*      * DCMS COMMAND *
*      ****
DCMS NAME IS DASDCMS SCALE IS DC02;
LOGON HEADING IS SYSTEM;
LOGOFF FOOTING IS SYSTEM;
PATHON HEADING IS SYSTEM;
PATHOFF FOOTING IS SYSTEM;
INPUT BUFFER TYPE IS BUF01;
OUTPUT MESSAGE CONTAINS 10 UNITS;
LOG LIMIT IS 1;
REPLY TIME LIMIT IS 5;
AUTHOR IS Y.NAGASAWA;
DATE-WRITTEN 82/06/03.

*
BUFFER TYPE IS BUF00;
UNIT IS MEMORY CONTAINS 128.
BUFFER TYPE IS BUF01;
UNIT IS MEMORY CONTAINS 256.
BUFFER TYPE IS BUF02;
UNIT IS MEMORY CONTAINS 512.
BUFFER TYPE IS SPF00;
UNIT IS MEMORY CONTAINS 200.

*
WORKSTATION TYPE IS WTBSC;
TRANSMISSION MODE IS BINARY;
SUPPRESS SYSTEM MESSAGE.
NODE IS LGBSC.
MEDIA IS INPUT;
DEVICE TYPE IS KB01.
MEDIA IS OUTPUT;
DEVICE IS PR01.
WORKSTATION TYPE IS WTTSSP;
TRANSMISSION MODE IS BINARY;
SUPPRESS SYSTEM MESSAGE.
NODE IS LGTSSP.
MEDIA IS INPUT;
DEVICE TYPE IS KB01.
MEDIA IS OUTPUT;
DEVICE IS PR01.
WORKSTATION TYPE IS WTTK;
TRANSMISSION MODE IS NONBINARY.
NODE IS TEKTR0.
MEDIA IS INPUT;
DEVICE TYPE IS KB01.
MEDIA IS OUTPUT;
DEVICE TYPE IS PR01.
WORKSTATION TYPE IS WTF6262R;
TRANSMISSION MODE IS NONBINARY.
NODE IS DPT01.
MEDIA IS INPUT;
DEVICE TYPE IS DP01.
MEDIA IS OUTPUT;
DEVICE TYPE IS DP01.
WORKSTATION TYPE IS WTFKANJI;
TRANSMISSION MODE IS NONBINARY.

NODE IS FKANJI.
MEDIA IS INPUT;
DEVICE TYPE IS DP01.
MEDIA IS OUTPUT;
DEVICE TYPE IS DP01.

```

システムバッファ定義

ワークステーションの定義

```

*
*
SCHEDULE TYPE IS SCHPH.
SCHEDULER IS BSCH FOR WTBSC;
MESSAGE SEGMENT IS TRANSMISSION BLOCK.
SCHEDULER IS BSCH FOR WTEK;
MESSAGE SEGMENT IS TRANSMISSION BLOCK.
SCHEDULER IS BSCH FOR WTTSP;
MESSAGE SEGMENT IS TRANSMISSION BLOCK;
HEADING IS 5.
SCHEDULER IS BSCH FOR WTF6262R;
MESSAGE SEGMENT IS TRANSMISSION BLOCK.
SCHEDULER IS BSCH FOR WTFKANJ1;
MESSAGE SEGMENT IS TRANSMISSION BLOCK.
*
END.

```

スケジュール型
の定義

ii) LD (論理宛先) の定義

```

LOGICAL ADDRESS COMMAND
LD GROUP NAME GDPLDG
SCALE IS LDG01;
CONNECTION LIMIT IS 20;
LDON LIMIT IS 20;
RESEND LIMIT IS 2.

LD TYPE IS USTLD;
STORAGE IS MEMORY.

LD NAME IS GULD01;
SERVICE IS PROGRAM.
RLD NAME IS GULD01.

LD NAME IS GULD02;
SERVICE IS PROGRAM.
RLD NAME IS GULD02.

LD NAME IS GULD03;
SERVICE IS PROGRAM.
RLD NAME IS GULD03.

LD NAME IS GULD04;
SERVICE IS PROGRAM.
RLD NAME IS GULD04.

LD TYPE IS GDPLD;
STORAGE IS MEMORY.

LD NAME IS GDPLD01;
SERVICE IS PROGRAM.
RLD NAME IS GDPLD01.

LD NAME IS GDPLD02;
SERVICE IS PROGRAM.
RLD NAME IS GDPLD02.

LD NAME IS GDPLD03;
SERVICE IS PROGRAM.
RLD NAME IS GDPLD03.

```

```

LD NAME IS G0PLD04;
SERVICE IS PROGRAM.
RD NAME IS G0PLD04.

END.

```

III) PROCEDUREの定義

```

*
* PROCEDURE COMMAND
*
PROCEDURE NAME IS G0PPROCI;
SCHEDULE TYPE IS SCHPH;
MESSAGE SCHEDULING MODE IS SYNCHRONOUS;
COMMUNICATION TYPE IS MULTI SESSION;
COMMAND TYPE IS CONNECTION WITH MESSAGE;
BUFFER TYPE IS INPUT MEMORY;
OUTPUT MEMORY;
ABEND MESSAGE IS SYSTEM.
*
*
*
MON NAME IS G0PMON1;
ON COMPLETION PASS ON;
LD GROUP IS DSLDG.
*
DD NAME IS G0PIN1 INPUT;
MESSAGE IS WITHIN 32767.
*
*
DO NAME IS G0POUT1 OUTPUT;
OPTION-CODE IS W C D F K.
*
SMON NAME IS G0MON10;
TYPE IS SHARE;
LD IS GULD01.
SMON NAME IS G0MON11;
TYPE IS SHARE;
LD IS G0PLD01.
*
*
*
END.

```

業務の動作環境設定

メッセージファイル)の定義
メッセージキューノード

IV) PEDの定義

```

*-----
*
*-----
* PED NAME IS GOPPED1;
*   SWAP POINT IS MESSAGE.
*-----
*
* AP NAME IS DUNNY;
*   TRANSACTION-POINT IS SYSTEM.
*-----
*
*-----
* DATABASE NAME IS LRSSDB7;
*   ACCESS MODE IS SHARE UPDATE.
*-----
*
*-----
* MON NAME IS GOPMON1 FOR GOPINI.
*-----
*
*-----
* MON NAME IS GOPMON3 FOR GOPNGT1.
*-----
*
* END.

```

} データベースの指定

} メッセージファイルの指定

B. LDPの起動方法

バッチジョブとして実行しているDMQPから、ローカルな問合せを処理する為に、LDPを起動することが必要である。DMQPでは、実行するLDPに対するジョブ制御文を生成する。この生成されたジョブ制御文に対して、これをインタナルリーダの入力に指定すると、バッチジョブとしてジョブキューに入れられ、実行される。LDPに渡されるローカル問合せは、ジョブ制御文内のSYSINのデータとして与えられる。

又、LDPでは、問合せに対して生成されるCOBOL DMLプログラムを実行する為のジョブ制御文が生成される。このCOBOL DMLプログラムの実行もインタナルリーダによって行なわれる。

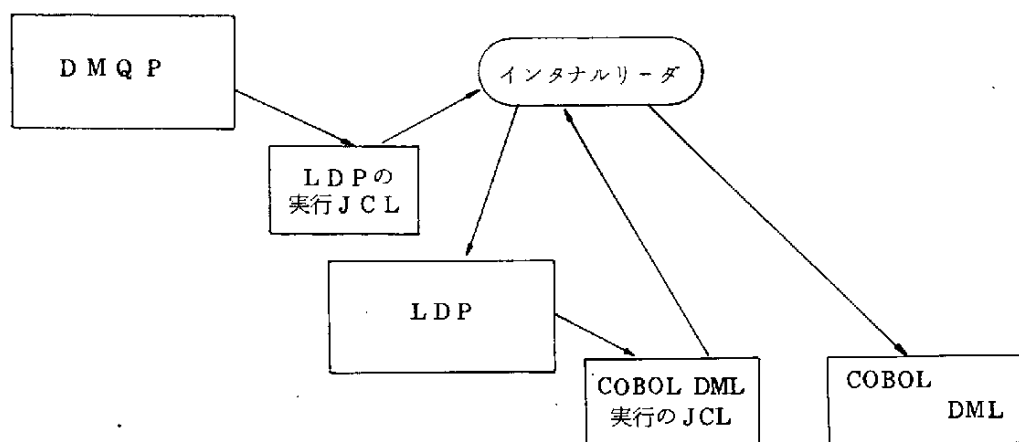


図 3.53 インタナルリーダによるLDP, COBOLの実行

【例】 DMQP から生成された LDP の実行用ジョブ制御文

```
//DCB0110B JOB 0110,JDDBS,REGION=1024K,MSGCLASS=X,NOTIFY=DCB0110
//JOB LIB DD DSN=GDP0110.LDPV35.LOAD,UNIT=SYSDA,VOL=SER=USER03,
// DISP=SHR
//E1 EXEC PGM=LDPV31T,PARM=BBBBBBBB
//SYSPRINT DD SYSOUT=X
//SYSOUT DD SYSOUT=X
//AIMPED DD DSN=AIM1.DRCTLB(LRSSPED7),DISP=SHR
//AIMDB001 DD DSN=AIM1.DB.LRSSPED7,DISP=SHR
//DMLFL DD DSN=DMLV5,UNIT=SYSDA,VOL=SER=WORK01,DISP=SHR
//CUBULFL DD DSN=CBBBBBB,UNIT=SYSDA,VOL=SER=WORK01,DISP=SHR
//JCLOUT DD SYSOUT=(A,INTRDR)
//JCLMST DD DSN=DCB0110.MMU1.CNTL(VBBEX),UNIT=SYSDA,
// VOL=SER=WORK01,DISP=SHR
//INTERF DD DSN=BBBBBBB,UNIT=SYSDA,VOL=SER=WORK01,DISP=SHR
//IDF DD DSN=BBBBBBB,UNIT=SYSDA,VOL=SER=WORK01,DISP=SHR
//SURLIB DD DSN=SYS1.SURLIB,DISP=SHR
//SURTIN DD DSN=BBBBBBB,UNIT=SYSDA,VOL=SER=WORK01,DISP=SHR
//SURTOUT DD DSN=BBBBBBB,UNIT=SYSDA,VOL=SER=WORK01,DISP=SHR
//SURTAK01 DD UNIT=SYSDA,SPACE=(CYL,(5))
//SURTAK02 DD UNIT=SYSDA,SPACE=(CYL,(5))
//SURTAK03 DD UNIT=SYSDA,SPACE=(CYL,(5))
//SRT1IN DD DSN=BBBBBBB,UNIT=SYSDA,VOL=SER=WORK01,DISP=SHR
//SRT1OUT DD DSN=BBBBBBB,UNIT=SYSDA,VOL=SER=WORK01,DISP=SHR
//SRT1AK01 DD UNIT=SYSDA,SPACE=(CYL,(5))
//SRT1AK02 DD UNIT=SYSDA,SPACE=(CYL,(5))
//SRT1AK03 DD UNIT=SYSDA,SPACE=(CYL,(5))
//SYSIN DD *
```

SYSIN の入力データとして、ローカル問合せが生成される。JCLOUT に COBOL DML プログラムを実行する為のジョブ制御文が生成され、インタナリータの入力となる。

C. BROADCAST の実現

AIM DB/DC 機能では、モジュール間の 1 対 1 通信は可能であるが、1 対 n 通信 (BROADCAST) は提供されない。DMQP は、他の DM の全ての DMQP に対してメッセージを BROADCAST する。この為、DMQP では、一対一通信を各 DM の DMQP に対して連続的に実行することで BROADCAST 機能を実現した。

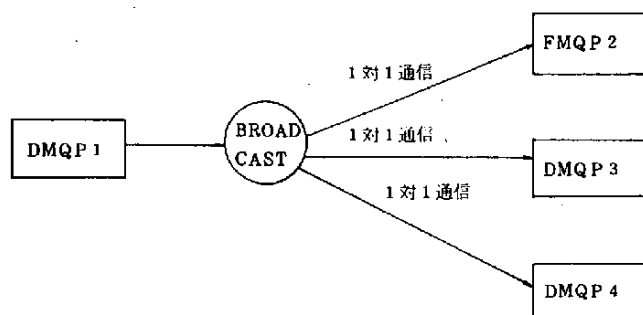


図 3.5 4 DMQP での BROADCAST の実現

3.3.6 DMQP間結合処理の実現

DMQP間結合処理では、ILQP後のGLQに対するDM間の結合式を処理する。ILQP後のGLQは、次の条件から成っている。

- (1) 結合述語としては、等価結合述語だけから成る。
- (2) 条件式は、次の様に正規化（積正規化）されている。

$$\text{qual} ::= C_1 \wedge C_2 \wedge \cdots \wedge C_n$$

$$C_i ::= X_{i1} \cdot a_{i1} \theta X_{i2} \cdot a_{i2} \quad (\theta \in \{=\})$$

DMQPは、broadcast 通信を使用したBBSアルゴリズムによって実現される。

BBSアルゴリズム

まず最初に、BBSアルゴリズムで参照する記号を定義する。

$$Q \triangleq \text{ILQP後のGLQ}$$

$$R_i \triangleq Q \text{が参照するILQPの結果関係}$$

$$\Sigma \triangleq \{R_1, \dots, R_n\} \ (n \geq 1) \triangleq Q \text{が参照する関係の集合}$$

$$A \triangleq Q \text{内の結合属性の集合}$$

(実際には、各属性の結合属性番号(jatt#)の集合である。結合属性は、この結合属性番号で参照される。)

$$TL \triangleq Q \text{内の目標属性の集合(jatt\#)}$$

$$AR_i \triangleq R_i \text{内の結合属性の集合}$$

$$TLR_i \triangleq R_i \text{内の目標属性の集合}$$

$$\text{dm}(R_i) \triangleq R_i \text{の存在するDM番号(DM\#)}$$

$$\Omega_{R_i} \triangleq R_i \text{の属性集合} = AR_i \cup TLR_i$$

$$SA \triangleq \text{single attribute joinで現在までに転送されたステージ属性の集合}$$

$$SAR_i \triangleq R_i \text{内のステージ属性} \\ (\text{Riが受信したステージ属性集合})$$

$$SR \triangleq \text{multi attribute現在までに転送されたステージ属性集合の集合}$$

$$SRR_i \triangleq R_i \text{が受信したステージ属性集合の集合}$$

$$\text{rf}(a) \triangleq a \text{を持っている関係の数}$$

BBSアルゴリズムの処理概要は、次の様である。

(0) [初期化]

全てのDMにおいて

$$\Sigma' \leftarrow \Sigma; \quad A' \leftarrow A; \quad \text{st\#} \leftarrow 0;$$

$SA \leftarrow SA_{R_i} \leftarrow \phi;$

(st#; ステージ番号)

$SR \leftarrow SR_{R_i} \leftarrow \phi;$

(1) [終了条件チェック]

if $A' = \phi$ and

全関係が ($\forall R \in \Sigma$) 同一の A_R を持ち、全ての属性 ($\forall a \in A_R$) がステージされている。

and

$|A| \geq 2$ のとき、少なくとも一つの関係が、multi-attribute join としてステージされている。

then go to (4)

(2) [ソース関係の決定]

各 DM_h で

a) $\forall a \in A'$, $\forall R_i \in \Sigma'$ の中で $|R_i[a]| = C_{ia}$ が最少の R_i 、 a を選択する。
ないとき、 $C_{ia} \leftarrow \text{MAX}$;

b) $\forall R_i \in \Sigma' (A_{R_i} = SR_{R_i})$ の中で、 $|R_i[\tilde{A}_{R_i}]| = \tilde{C}_i$ が最少の R_i を選択する。
ないとき、 $\tilde{C}_i \leftarrow \text{MAX}$; $R_i[\tilde{A}_{R_i}]$ は multi-attribute join のソース関係を表わす。

C_i はそのカーディナリティである。

c) $st\# \leftarrow st\# + 1$;

if $C_{ia} > \tilde{C}_i$

then MAJP [R_i]; {multi-attribute join 処理}

else SAJP [R_i, a]; {single-attribute join 処理}

(3) [bif-map response 処理]

BMJP [st#];

(4) [終了]

{結果関係をRDMに生成}

- $\forall R_i \in \Sigma$ に対して R_i を有する DM_i は RDM に R_i を broadcast する。
- RDM は broadcast されてくる R_i を結合して結果関係を作る。

A. SAJP の実現

SAJP は、single attribute 結合処理を行う。関係 R_i の属性 a の射影 $R_{ia} (= R_i[a])$ の broadcast と、各 DM_j での bit-map response broadcasting と bit-map response 処理を行う。

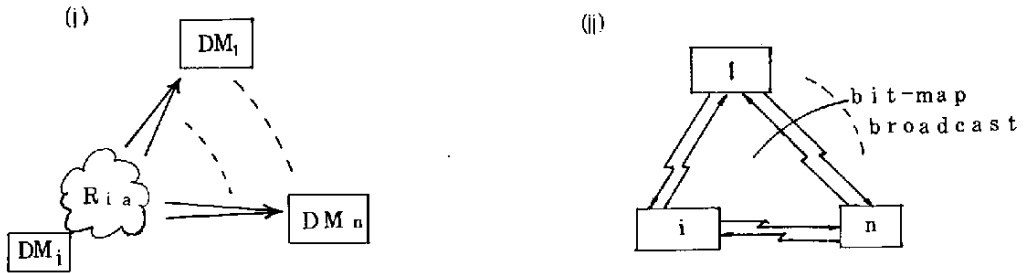


図 3.5.5 SAJP の処理概要

SAJP の処理は、次の様に行なわれる。

(1) 各DMで、

```

 $A' \leftarrow A' - \{a\}; \quad RM \leftarrow \phi;$ 
 $SA \leftarrow SA \cup \{a\}; \quad st\# \leftarrow st\# + 1; \quad (stage\#)$ 
 $\forall R_j \in \Sigma$  に対し
    if  $a \in AR_j$  then
        do ;
             $SAR_j \leftarrow SAR_j \cup \{a\};$ 
             $RM \leftarrow RM \cup \{R_j\};$ 
        end ;

```

(2) ソース関係の broadcast

```

 $DM_i$  は  $R_{ia} \leftarrow R_i[a];$ 
        broadcast  $R_{ia};$  go to (4) ;

```

(3) 結合処理と bit-map response

$a \in SAR_j$ なる各 DM_j は R_{ia} を待つ。

(i) R_{ia} を receive したならば

```

 $R_j \leftarrow R_j, [a=a]R_{ia}; \quad [join];$ 
 $ACK_j \leftarrow \phi;$ 

```

(ii) 各 staged join attribute $b \in SAR_j - \{a\}$ に対して

```

 $BM_{jb} \leftarrow R_j[b]$  の  $R_{jb}$  に対する bit-map ;
 $ACK_j \leftarrow ACK_j \cup \{(card(R_j[b]), BM_{jb})\};$ 

```

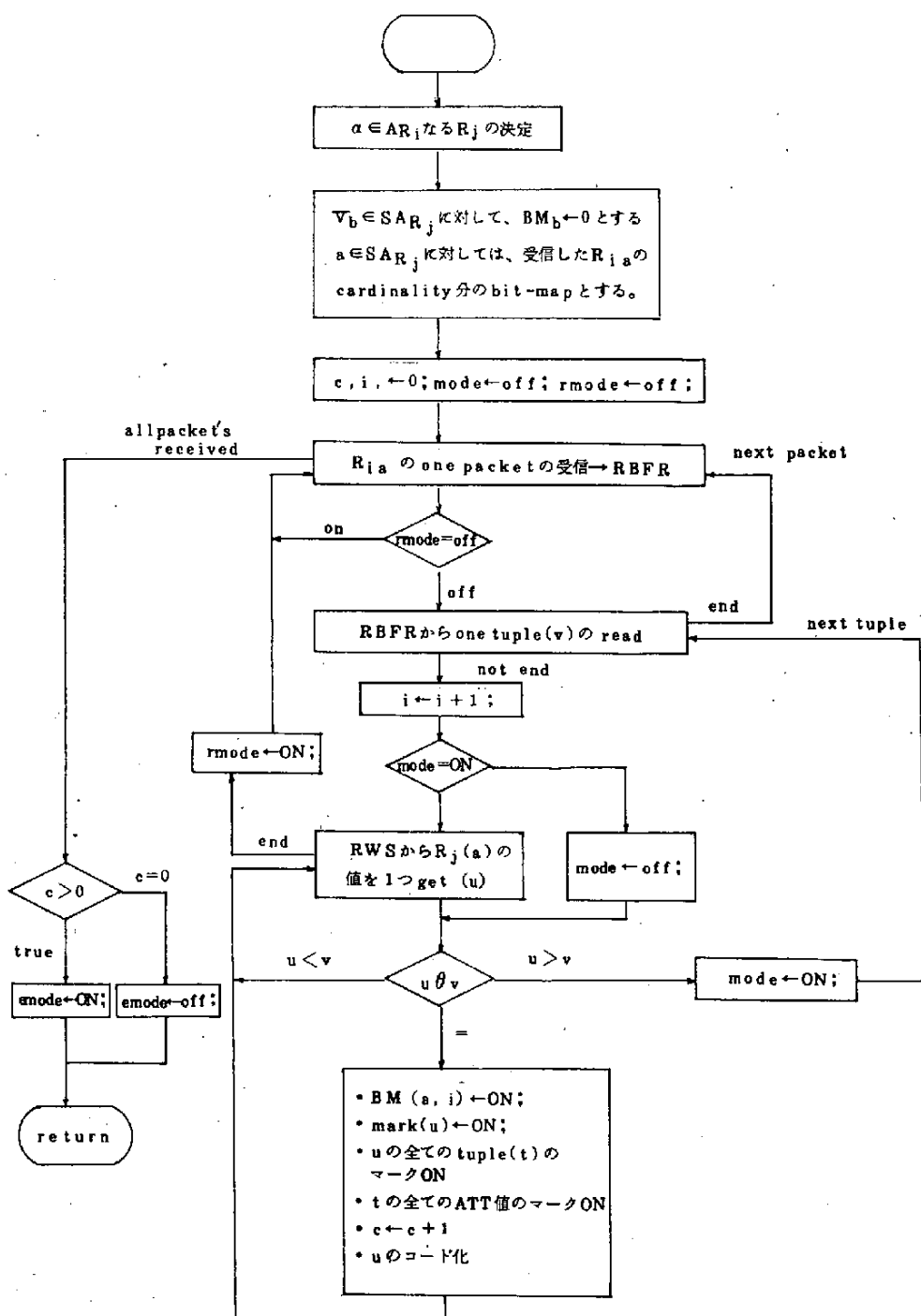
stage attribute a に対して

$$\begin{cases} \text{BM}_{ja} \leftarrow R_j[a] \text{ の } R_{ja} \text{ に対する bit-map ;} \\ \text{ACK}_j \leftarrow \text{ACK}_j \cup \{(\text{card}(R_j[a]), \text{BM}_{ja})\} \end{cases}$$

(iii) broadcast ACK_j ;

結合処理 (SAJOIN処理) の処理フローは図 3.5 6 様である。

(4) return;



B. MAJPの実現

MAJPは、multi-attribute 結合処理を行う。コード化された $R_i[\tilde{A}_{R_i}]$ を broadcast して、 $A_{R_i} \cap A_{R_j} = \phi$ ($i \neq j$) なる R_j と coded join を行なう。

DM_j はこの結果を bit-map として broadcast する。

MAJPは、次の様に行なわれる。

(1) [初期化]

$$\Sigma' \leftarrow \Sigma' - \{R_i\};$$

$$SR \leftarrow SR \cup \{R_i\}; \quad RM \leftarrow \phi;$$

$$\forall R_j \in \Sigma (A_{R_j} \cap A_{R_i} \neq \phi) \text{ について}$$

$$SR_j \leftarrow SR_j \cup \{R_i\};$$

$$A_{ij} \leftarrow A_{R_j} \cap A_{R_i};$$

(2) [コード化]

$\forall R_j \in \Sigma (A_{ij} \neq \phi)$ に対して、 $\forall a \in A_{ij}$ をコード化する。

$$\text{CODE } [R_j];$$

$$RM \leftarrow RM \cup \{R_j\};$$

(3) [ソース関係 $R_i[\tilde{A}_{R_i}]$ の broadcast]

$$DM_i \text{ は } \tilde{R}_i \leftarrow R_i[\tilde{A}_{R_i}];$$

$$\text{broadcast } \tilde{R}_i; \quad \text{go to (5)};$$

(4) [結合処理と bit-map response の broadcast]

各 DM_j ($A_{R_j} \cap A_{R_i} (= A_{ij}) \neq \phi$) に対して

[coded join]

$$R_j \leftarrow R_j [\tilde{A}_{ij} = \tilde{A}_{R_i}] R_j; \quad \text{ACK}_j \leftarrow \phi_j;$$

$$RBM_{ji} \leftarrow R_j[\tilde{A}_{R_i}] \parallel \tilde{R}_i; \quad \text{ACK}_j \leftarrow \text{ACK}_j \cup \{RBM_{ji}\};$$

[RBM生成]

$\forall R_h \in SR_{R_j} - \{R_i\}$ に対して

$$A_{Rh} \leq A_{R_i} \text{ ならば } SR_{R_j} \leftarrow SR_{R_j} - \{R_h\};$$

$$\text{otherwise } RBM_{jh} \leftarrow R_j[\tilde{A}_{R_h}] \parallel R_h;$$

BM_{jh} の bit が 1 つでも off なら

$ACK_j \leftarrow ACK_j \cup \{RBM_{jh}\};$

[BM生成]

$\forall_b \in SAR_j - A_{ji}$ (コード化されていない staged attribute)
に対して

$BM_{jb} \leftarrow R_j[b] \parallel R_{jb};$

if BM_{jb} is all bit on,

else $ACK_j \leftarrow ACK_j \cup \{BM_{jb}\};$

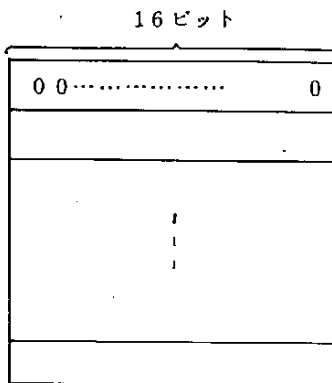
[broadcast of bit-map response]

broadcast $ACK_j;$

(5) return;

C. BIT-MAPの管理

結合処理の結果に対する bit-map は、DMQP で次の様に管理される。bit-map は、16 bits を基本単位とする図 3.57 の様な bit-map buffer に格納される。結合処理の結果は、組のカーディナリティ分のビットに対して、条件を満たす組の場合はビットがオン、満たさない場合はビットがオフになっている。結果の



bit-map は、bit-map buffer に 16 ビット単位で格納され、余ったビットはオフにする。例えば、結果のカーディナリティが 30 の場合、bit-map buffer の基本単位が 2 つ (32 ビット) 使用され、後ろの 2 ビットがオフになる。

bit-map buffer には、すでに実行された結合処理結果の bit-map が全て格納される。結合処理に対する bit-map を得る為に、図 3.58 の様な管理テーブルを設けている。

図 3.57 bit-map buffer

relid	DQRAT#	jatt#	st#	ED#

図 3.58 bit-map管理テーブル

rel-id は関係識別番号、jatt# は結合属性番号である。この組で、single attribute joinの結果ビットマップを一意に識別する。ST#、ED# は各々 bit-map buffer 内の bit-map の開始位置、終了位置を表わしている。

3.3.7 GCS 問合せの実行

GCS 問合せ (GQ) の実行は、UST と DMQP がお互いにメッセージを通信し、協同して行なう。実行は、次の様に行なわれる。

- 1) GQ を表現変換し、Local GCS 問合せを生成する。
- 2) Local GCS 問合せから G-tree を生成する。

G-tree は、図 3.59 の様な構造である。

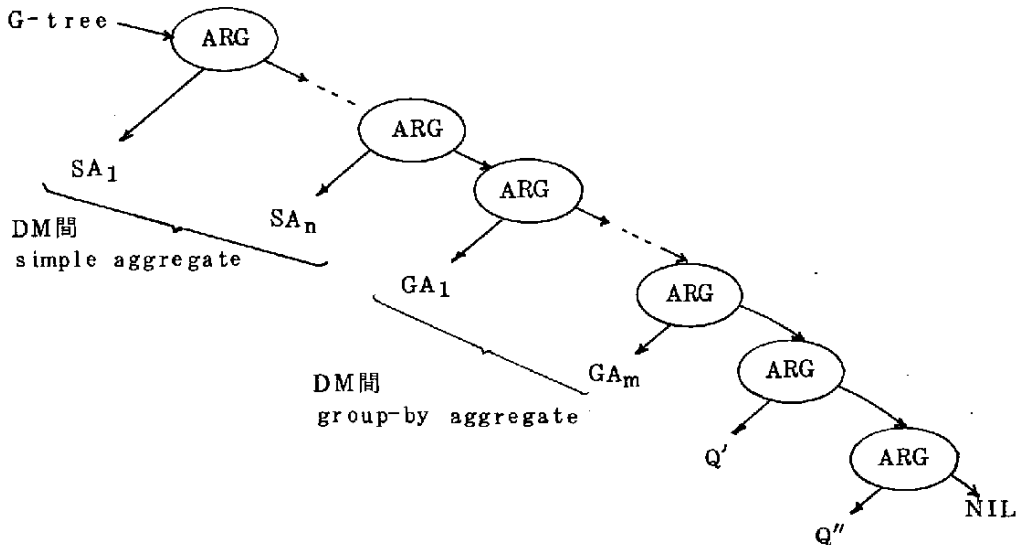


図 3.59 G - tree

- 3) 各 SA_i , GA_j , Q' , Q'' に対して必要な Q -tree を生成する。
- 4) 問合せ分割 (QD) によって $\{ILQ_i\}$ 及び DMQ を生成する。 $\{ILQ_i\}$ は、ローカル問合せである。 DMQ は、DM間にまたがる問合せである。これを、 $ILQP$ メッセージとして、 $DMQP$ に送信する。
- 5) $DMQP$ は、 $ILQP$ メッセージを受信し、他の $DMQP$ と通信処理を行なう。自DM内のDBSへのローカル問合せ (LQ) があれば、 LDP を起動する。全ての通信処理が終了したら、結果を関係作業領域 (RWS) に格納し、終了を UST に通知する。
- 6) $DMQP$ からの結果に対して必要な aggregate 処理を行ない、結果を利用者に出力する。
 UST は、5) を除く 1) ~ 6) までの処理を行なう。 $DMQP$ は、5) の処理を行なう。
 GCS 問合せの入力から、結果を利用者に出力するまでの処理概要を図 3.60 に示す。

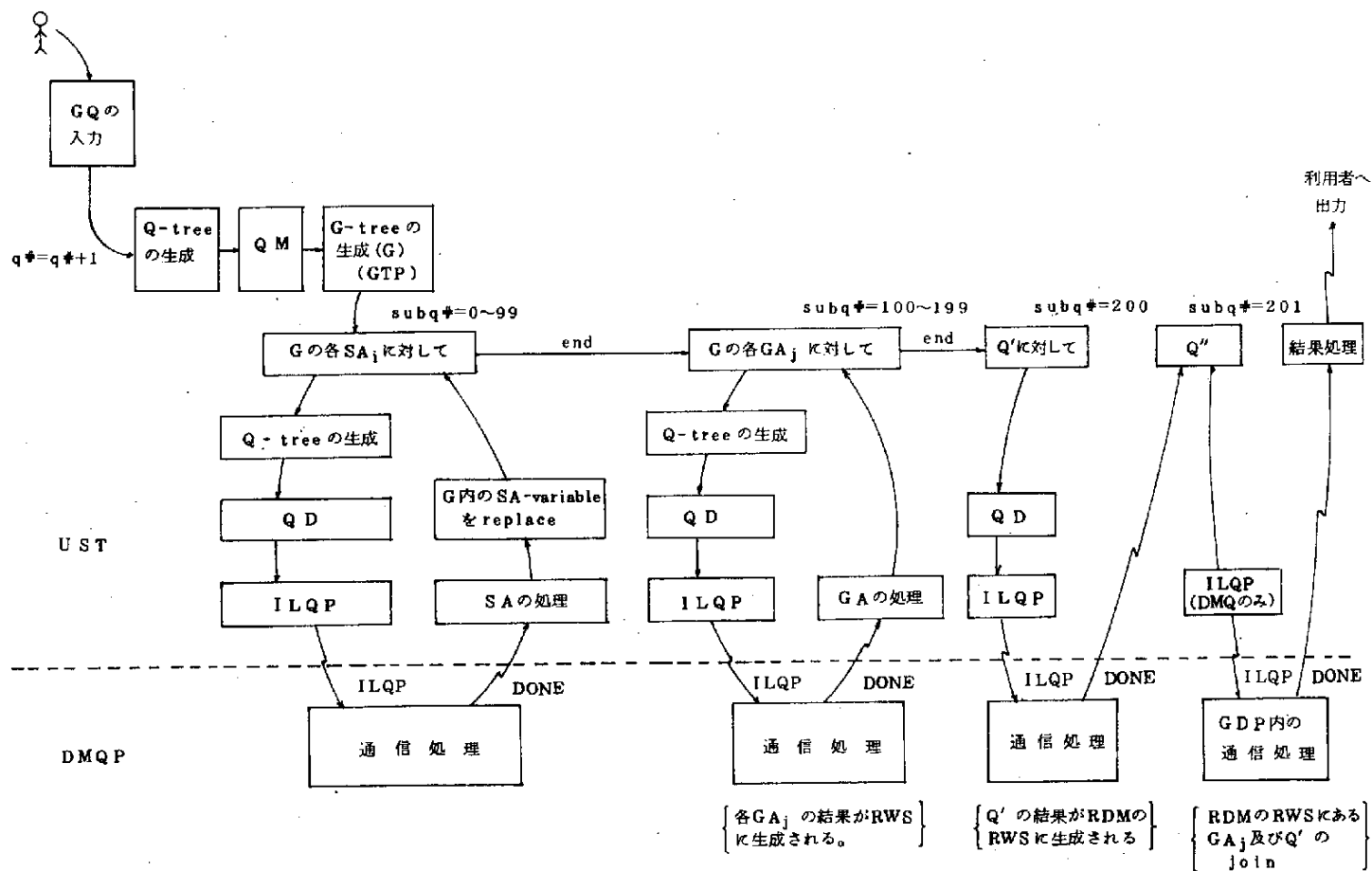


図 3.6 0 GCS 問合せの処理概要

4. J D D B S の実働例

J D D B S の利用者としては、管理者と一般利用者との大別される [図 4.1]。管理者は、D I の L C S 情報生成及び G C S 定義の D I への格納を行なう。一般利用者は、管理者の定義した G C S を参照する G C S 問合せを入力する。

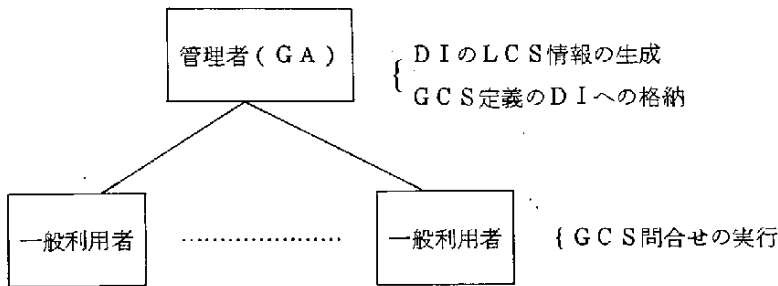


図 4.1

J D D B S を実働させる為には、まず最初に D I の L C S 情報を生成する。次に D I P に G C S を定義し、D I を生成しなければならない。D I 内に定義されている G C S に対して一般利用者は R Q L による G C S 問合せを記述し、J D D B S に実行させる事が出来る。

A J D D B S の実現イメージ

J D D B S の実現イメージを以下に示す。

- (1) M-170F内に、4つのDMを作成する。
- (2) 各DBSは、各々異なるdisk上に作成する。
- (3) DBSとしては、
 - (i) PRDBSを3つコピーする。各コピーに対して、異ったサブスキーマを設ける。
(PRDBS1~PRDBS3)
PRDBSは、プロジェクト管理データベースである。
 - (ii) DCDBSを設ける。DCDBSは、図書管理データベースである。
- (4) GDP-GDP通信は、FACOM OSN提供のプロセス間通信(AIM DB/DC)を用いて実現する。
- (5) GDP-LDP通信は、FACOM OSN提供のインタナルリーダを用いて実現する。
- (6) HI、RWS、DIは、AIMを用いて同一DBとして実現する。

図 4.2 に J D D B S の実現イメージを示す。各DM内のDBS (PRDBS1~PRDBS3、DCDBM) のLIS及びLCSを以下に示す。

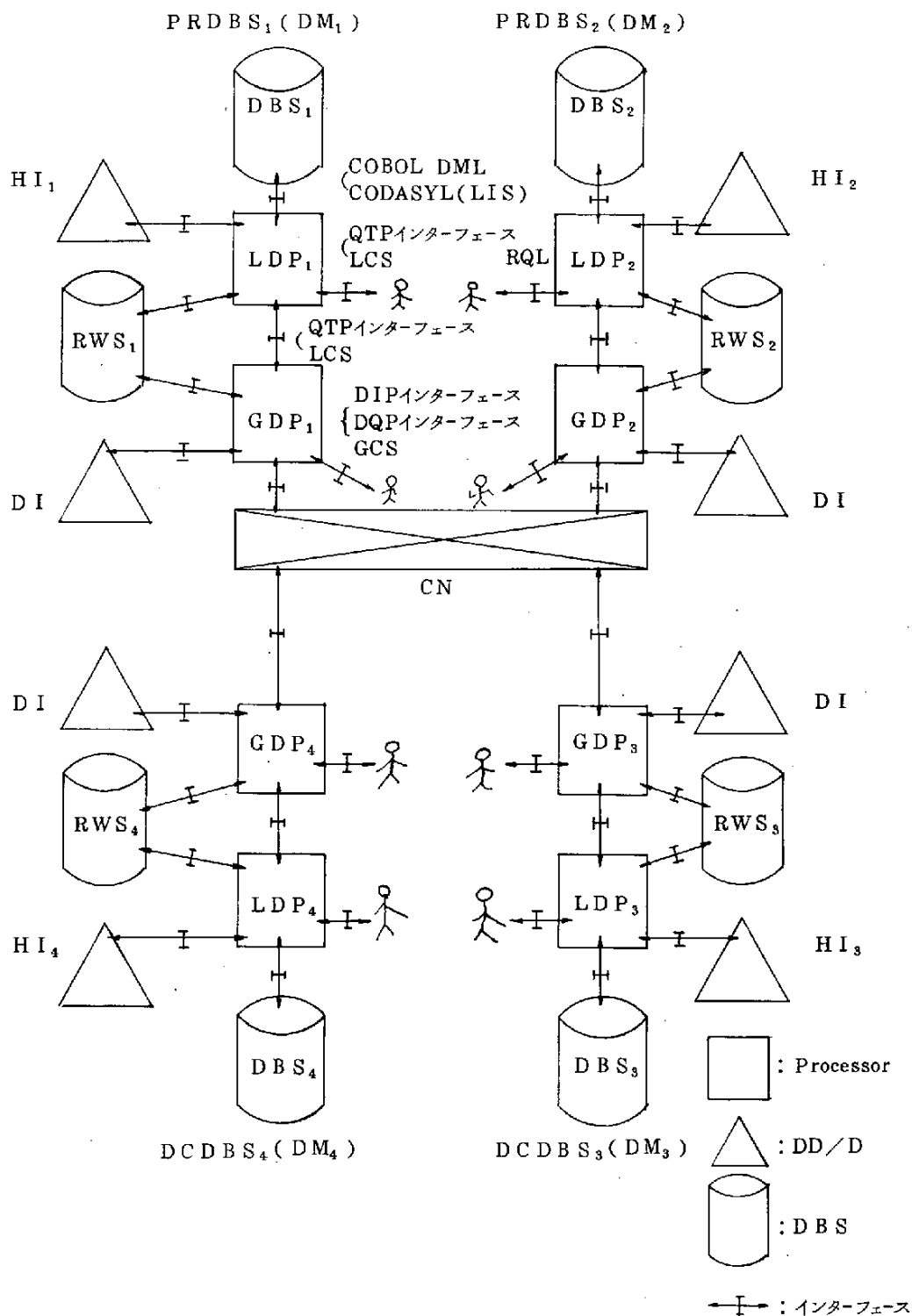


図 4.2 JDDBSの実現イメージ

a. PRDBS 1

PRDBS 1は、従業員とプロジェクトとの関係を表わすデータベースである。PRDBS 1のLISとLCSを図4.3に示す。

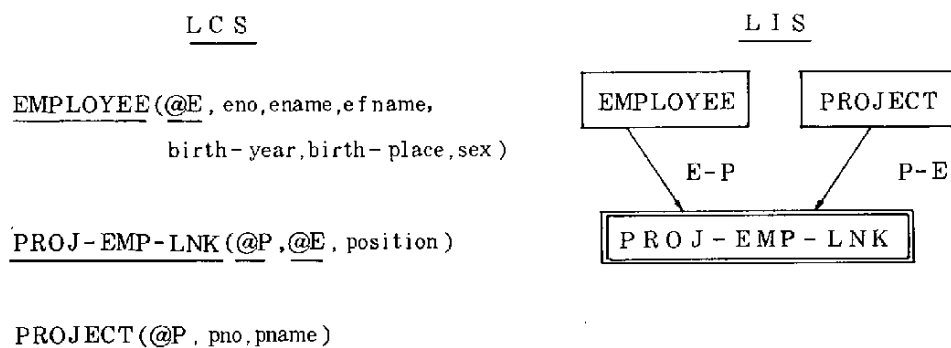


図4.3 PRDBS 1のLISとLCS

b. PRDBS 2

PRDBS 2は、主題とプロジェクト、及びそのプロダクトとの関係を表わすデータベースである。PRDBS 2のLISとLCSを図4.4に示す。

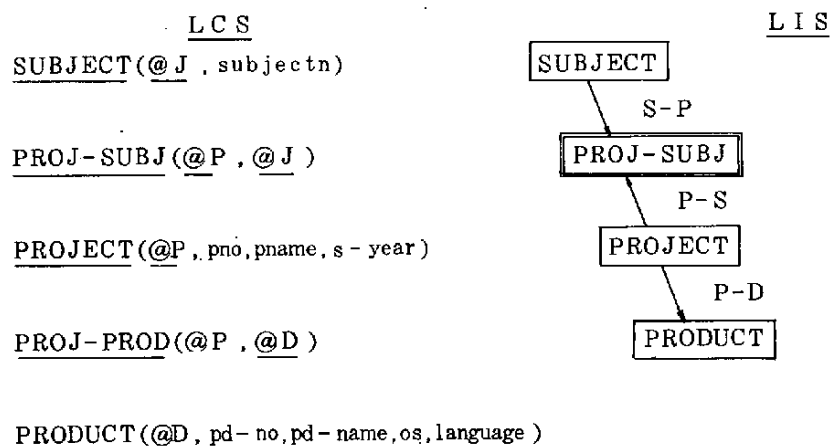


図4.4 PRDBS 2のLISとLCS

c. PRDBS 3

PRDBS 3は、従業員、レポートと主題間の関係を表わすデータベースである。PRDBS 3のLISとLCSを図4.5に示す。

LCS

SUBJECT (@J , subjectn)

REP-SUBJ (@R , @J)

REPORTR (@R , rno , rname ,
cost , pyear , page)

EMP-REP-LNK (@E , @R ,
auth - no)

EMPLOYEE (@E , eno , ename ,
efname , birth - place)

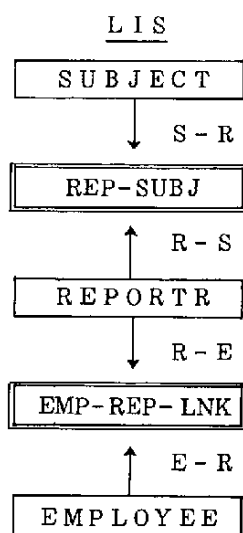


図 4.5 PRDBS3のLISとLCS

d. DCDBS

DCDBSは、図書管理データベースである。DCDBSのLISとLCSは1.1.2を参照。

B. DIPの起動方法

DIPはM-170F (OS / IV F4) TSSの利用者プログラムとして実行される。DIPを起動し、GCSを定義し、DIに格納する手順を次に示す。例題内の下線は、利用者が入力するコマンドを示す。その他はシステムの表示である。

(i) TSSセッションの開始

LOGON TSS DCB0110/DCB
ENTER PROCEDURE NAME -
GDPV3

(ii) DIPの起動

EXEC MMUI(DIPV3T)

PLEASE LOGON

(iii) DIPのログオン

LOGON GDP SUZUKI/MAKO;

↑ ↑

利用者識別名 パスワード

(iv) GCS名の宣言

```
/00010  DEFINE GCSNAME11;
```

GCS名

(v) GCS関係定義(DIPコマンド)

```
{ #99999  <DIP-Command>; }
```

(vi) DIへの格納

```
#00180  STORE;
```

(vii) DIPの終了

```
/99999  LOGOFF;
```

C. DIPの実働例

JDDBS内のDBS (PRDBS1~3、DCDBS) に対して図4.6の様なGCSを定義する場合のDIP実例を以下に示す。

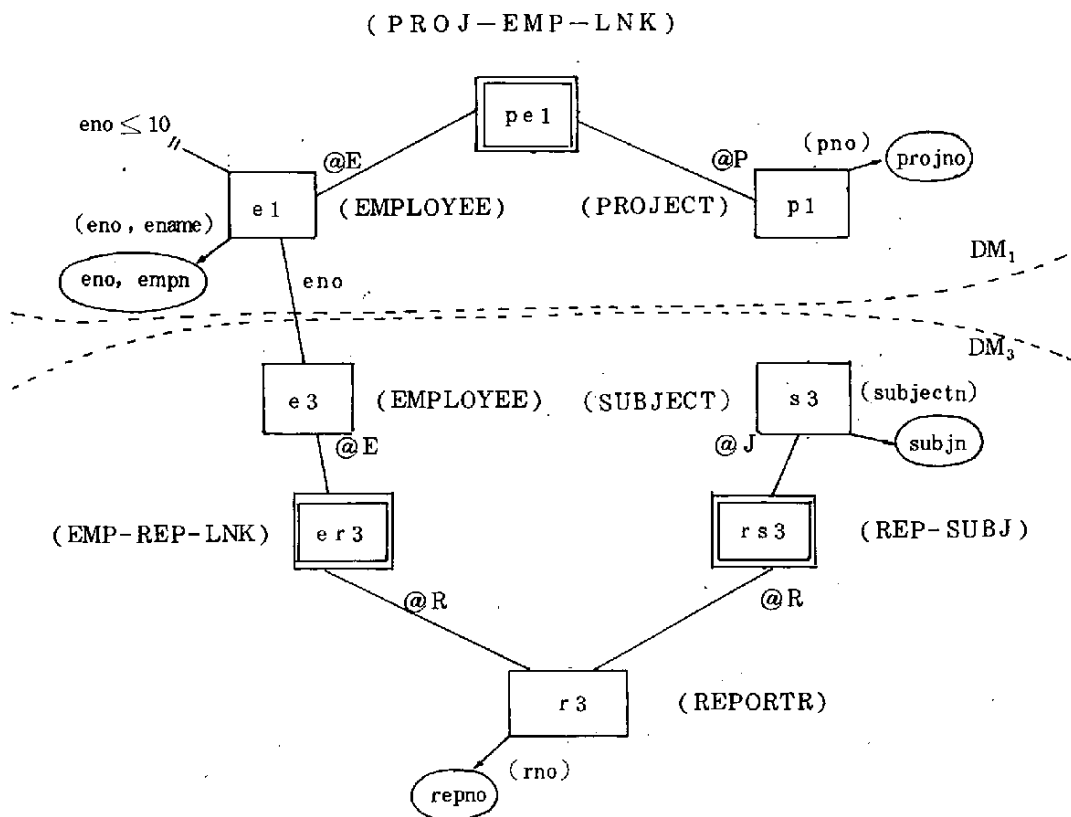


図4.6 GCSNAME11 (GREL11) の定義

EXEC MMUI(DIPV3T)

PLEASE LOGON

LOGON GDP SUZUKI/MAKO;

/00010 DEFINE GCSNAME11;

#00010 RANGE (E1, EMPLOYEE :1);

#00020 RANGE (PE1, PROJ-EMP-LNK :1);

#00030 RANGE (P1, PROJECT :1);

#00040 RANGE (S3, SUBJECT :3);

#00050 RANGE (RS3, REP-SUBJ :3);

#00060 RANGE (R3, REPORTR :3);

#00070 RANGE (FR3, EMP-REP-LNK :3);

#00080 RANGE (E3, EMPLOYEE :3);

#00090 DEFINE GREI11 (ENO/ENO/15, EMPN/ENAME/C16, PROJNO/PNO/C8,

SUBJIN/SUBJECTIN/C100, REPNO/RNO/C8)

#00110 (ENO=E1.ENO, EMPN=E1.ENAME, PROJNO=P1.PNO,

SUBJIN=S3.SUBJECTIN, REPNO=R3.RNO)

#00130 WHERE

#00140 P1.OP = PE1.OP AND PE1.OE = E1.OE AND

#00150 S3.OJ = RS3.OJ AND RS3.OR = R3.OR AND

#00160 R3.OR = FR3.OR AND FR3.OE = E3.OE AND

#00170 E1.ENO = E3.ENO AND E1.ENO <= 10;

#00180 STORE;

D. USTの起動方法

USTはTSSのもとで実行される。USTを起動し、GCS問合せを作成し、実行させ結果を得る。各DM内のDMQPは、バッチジョブとして起動しておく。各DM内のDMQPを起動するジョブ名を表4.1に示す。

USTの起動及びGCS問合せの手順を以下に示す。

(i) TSSセッションの開始

LOGON TSS DCB0110/DCB

ENTER PROCEDURE NAME -

UST01

(ii) USTの起動

EXEC MMUI(USTIV3A)

PLEASE LOGON

起動するプログラム名はセッション開始時のプロシジャー名と対応させなければならない。対応表4.2に示す。

表 4.1 DMQP のジョブ名

DM	DMQP のジョブ名
1	GDP JCL A
2	GDP JCL B
3	GDP JCL C
4	GDP JCL D

表 4.2 プロシジャー名と
応用プログラム名の対応表

DM	プロシジャー名	応用プログラム名
1	UST01	USTV3A
2	UST02	USTV3B
3	UST03	USTV3C
4	UST04	USTV3D

(iii) UST のログオン

LOGON GDP SUZUKI/MAKO;

↑ ↑
利用者識別名 パスワード

(iv) 参照する GCS 名の宣言

/00010 DQP GCSNAME11;

(v) GCS 問合せの実行 (DQP コマンド)

{ #99999 <DQP-command>; }

(vi) UST の終了

/99999 LOGOFF;

E. UST の実働例

GCSNAME11 内の GREL11 関係に対する問合せの実働例を次に示す。問合せは、レポート番号が "TAKIM82" より大きいレポート番号を持つ従業員の名前 (EMPNO)、レポート番号 (REPNO)、プロジェクト番号 (PROJNO) を求めるものである。

EXEC MMUI(USTV3A)

```
PLEASE LOGON
LOGON GDP SUZUKI/MAKO;
/00010 DQP GCSNAME11;
#00010 RANGE (R11, GREL11);
#00020 RETRIEVE INTO TEMP (R11.EMPNO, R11.REPNO, R11.PROJNO)
#00030 WHERE
#00040 R11.REPNO > "TAKIM82";
#00050 GO;
```

THE ELAPSE TIME FOR RETRVP,QM.. 67

⋮

GCS 問合せ TEMP は、図 4.7 の様な問合せである。

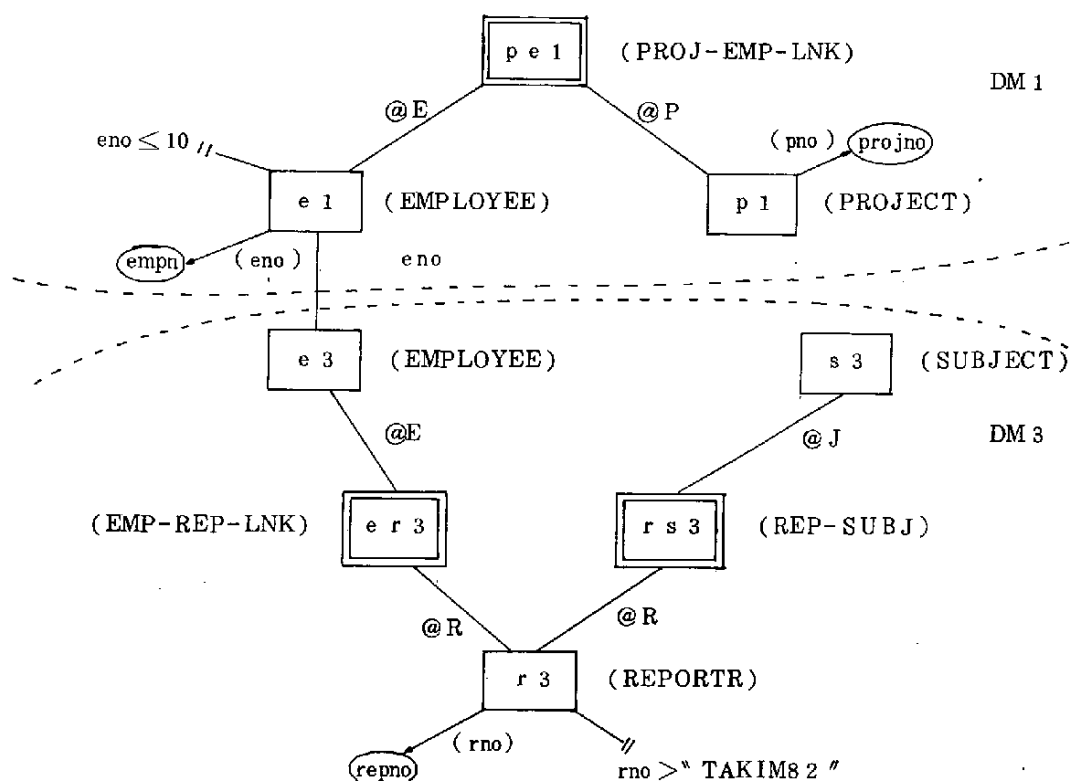


図 4.7 問合せTEMP

GCS問合せTEMPは、問合せ分割 (QD) によって、次の2つのローカル問合せ (LQ) とDM間問合せ (DMQ) を生成する。

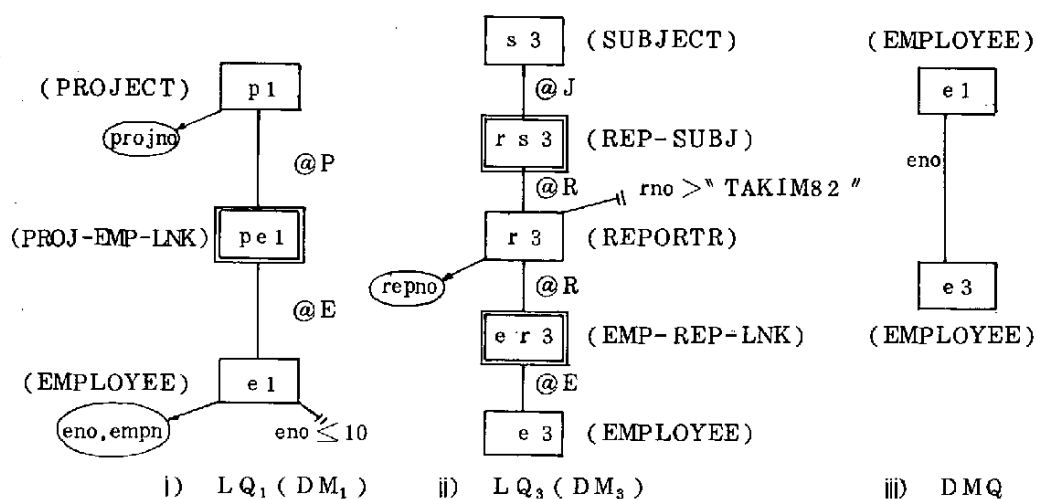


図 4.8 LQとDMQ

これを ILQP メッセージとして、自 DM 内の DMQP へ送る。

ILQP メッセージを受けた DMQP は、各 DM 内の DMQP に ILQP メッセージを送る (図 4.9)。

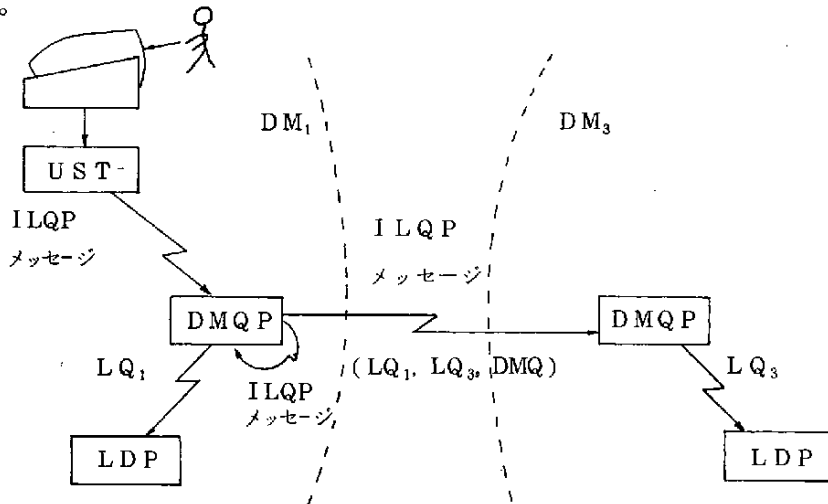


図 4.9 DMQP への ILQP メッセージ送出

各 DM 内の DMQP は、ローカル問合せ (LQ) を実行する LDP を起動する。例題の場合、ローカル問合せ (LQ) が実行されるのは、DM1 と DM3 である。

DM1 では、次の様な LQ1 を生成し、これを実行する LDP が起動される。

```
----- SUBMIT_LDP -----
RANGE (V001, EMPLOYEE      );
RANGE (V002, PROJ-EMP-LNK   );
RANGE (V003, PROJECT       );
RETRIEVE INTO RREL010051 (TL001=V001.ENAME , TL002=V001.ENU , TL003=V003.PNO )
WHERE V003.EMPNO=V002.EMPNO AND V001.ENO <= 10 AND V002.EMPNO=V001.EMPNO;
```

この LQ をデータベース PRDBS1 で実行し、次の様な結果を得る。

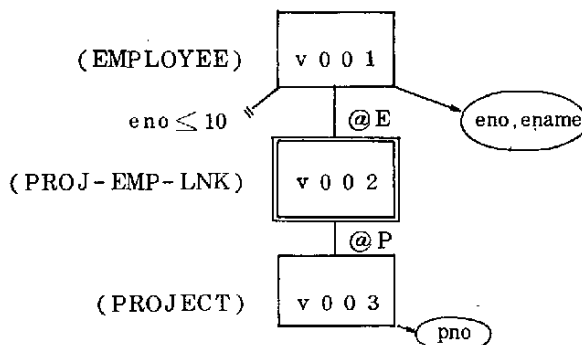


図 4.10 DM1 内の LQ1

LENAME	LENO	LPNO	
IAZUMA	1	91C858	1
IAZUMA	1	91JIPNET731	
IAZUMA	1	91RSS75	1
IHAMANAKA	1	31DDBS80	1
IHAMANAKA	1	31KIS81	1
IHAMANAKA	1	31REK81	1
IITO	1	81C858	1
IITO	1	81DDBS80	1
IITO	1	81JIPNET731	
IKAJI	1	71JIPNET731	
IKAJI	1	71INTT81A	1
IKAJI	1	71RSS81B	1
IKAJI	1	71SCRTY	1
IKAJI	1	715G79	1
IKOIKE	1	21C858	1
IKOIKE	1	21DDBS80	1
INISHIHARA	101	1JIPNET731	
INISHIHARA	101	1RSS75	1
INOMURA	1	41DDBS80	1
INOMURA	1	41KIS81	1
IOGAWA	1	61JIPNET731	
IOGAWA	1	61RSS75	1
IOGAWA	1	615G79	1
IOGAWA	1	615G80	1
ITAKIZAWA	1	11DDBS80	1
ITAKIZAWA	1	11JIPNET731	
ITAKIZAWA	1	11MMUI80	1
IYAMAMOTO	1	51JIPNET731	
IYAMAMOTO	1	51SCRTY	1
IYAMAMOTO	1	515G79	1
IYAMAMOTO	1	515G80	1

図 4.11 LQ₁の実行結果

DM 3では、ローカル問合せとして次のLQ 3が生成され、これを実行するLDPが起動される。

```

----- SUBMIT_LDP -----
RANGE (V004, SUBJECT          );
RANGE (V005, REP-SUBJ         );
RANGE (V006, REPORTR          );
RANGE (V007, EMP-REP-LNK      );
RANGE (V008, EMPLOYEE         );
RETRIEVE INTO RRELO30052 (TL001=V006.RNO , TL002=V008.END ) WHERE V00
7.@E = V008.@E AND V006.RNO > "TAKIM82" AND V006.@R = V007.@R AN
D V005.@R = V006.@R AND V004.@J = V005.@J ;

```

IRNO	ENR
TAKIM82A	1
TAKIM82B	1
TAKIM82B	14
TAKIM82B	21
TAKIM82C	1
TAKIM82C	21
TANAY76A	24
TANAY76A	68
TANAY76A	69
TANAY76A	70
TANAY76A	71
TANAY77A	24
TANAY77A	71
TANAY79A	24
TANAY79B	24
TANAY79B	71
TANAY80A	24
TANAY80A	47
TANAY80A	48
TANAY81A	1
TANAY81A	24
TANAY81B	24
TANAY82A	24
WONGE76A	52
WONGE76A	67
WONGE77A	52
WONGE79A	52
YAMAK75	5
YAOS 77A	50
YAOS 77B	50
YAOS 78A	50
YAOS 78A	51
YAOS 78A	59
YAOS 78A	77
YAOS 78A	78
YAOS 78B	50
YAOS 78B	79
YAOS 79A	50
YAOS 79B	50
YAOS 80A	50
YOKOM81	21
YOKOM82A	1
YOKOM82A	14
YOKOM82A	21

図 4.13 LQ₃の実行結果

このLQ₃を、DM3内のデータベースPRDBS3上で実行し、次の様な結果を得る。

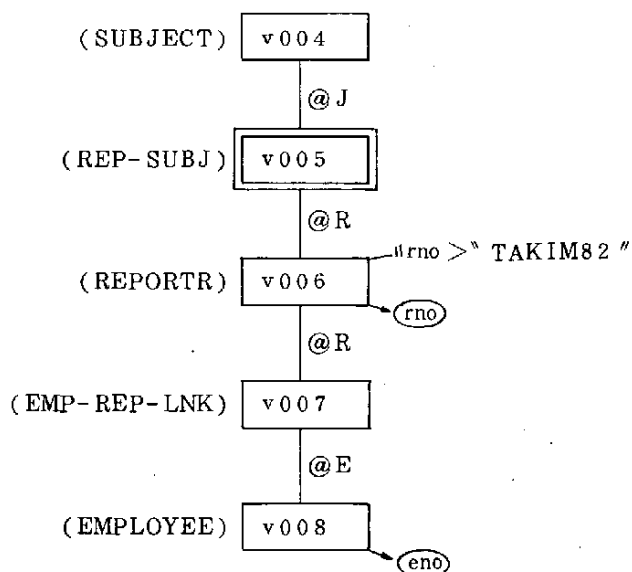


図 4.12 DM3内のLQ₃

各DMのローカル問合せの実行が終了すると、LQの実行結果の組数の情報 (ILQP-ACKメッセージ) を他のDMQPへ送る。

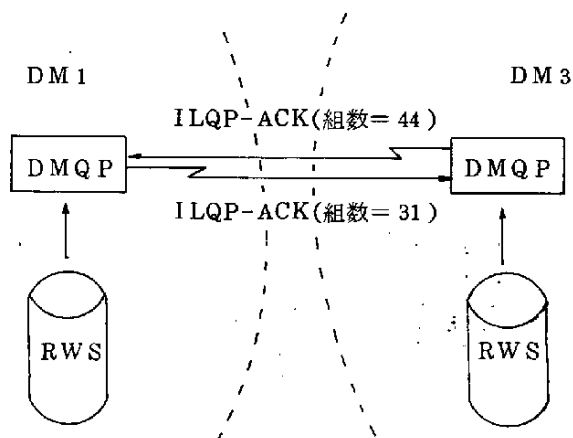


図 4.14 ILQP-ACKメッセージの送信

次にDM間問合せ (DMQ) を実行する。例題の場合、 LQ_1 、 LQ_3 の結合属性 eno の結合処理を行なう。この場合、 LQ_1 の結果の方が eno の値が少ないので、DM1 から結合属性 eno の値が送られる (図 4.15)。

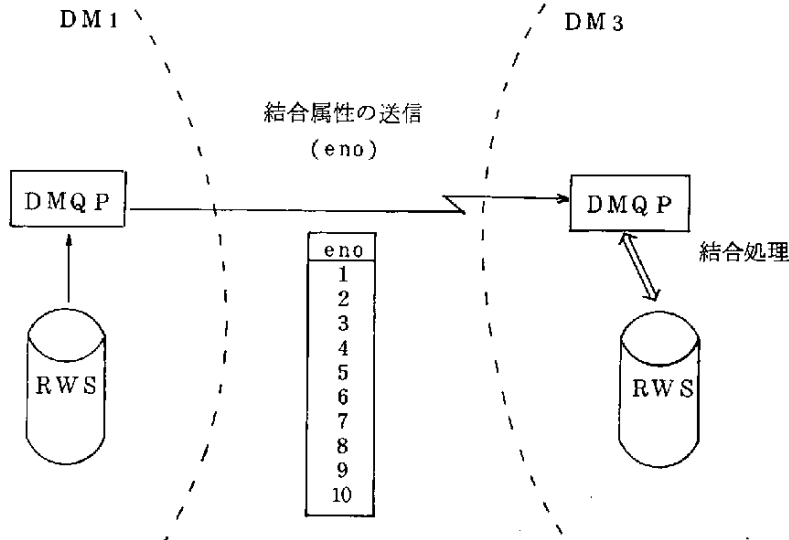


図 4.15 結合属性の送信

DM3 では、送られてきた結合属性 (eno) の値と LQ_3 の結果属性 (eno) の値とで結合処理を行なう。各属性の値とも昇順にソートされている。結合処理の結果は、送られてきた結果属性の組数分のビットに対して、条件を満たす値はビットオン (=1)、条件を満たさない値はビットオフ (=0) が各々セットされる。このビットマップを DM1 に送る (図 4.16)。結合条件を満たさない組は RWS から削除される。

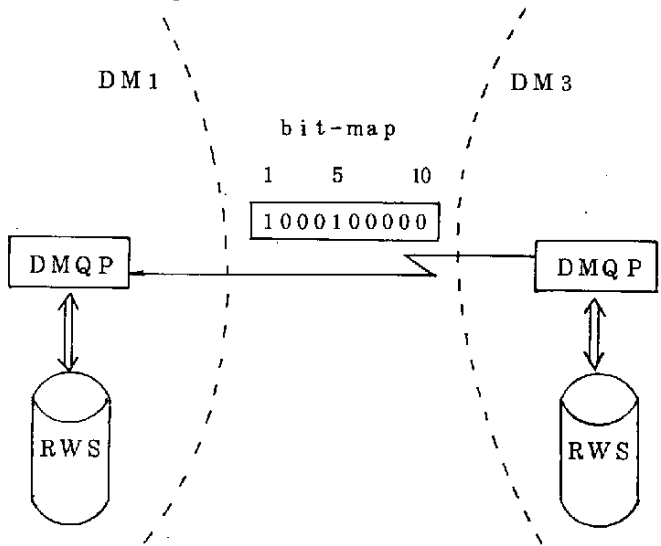


図 4.16 bit-map の送信

結合条件を満たす組だけが、各々のRWSに残される。この例で、結合条件を満たす図4.17の組が各々のRWSに残される。

ENAME	ENO	PNO		IRNO	IENO	
ITAKIZAWA	I	11DDBS80	I	ITAKIM82A	I	11
ITAKIZAWA	I	11JIPNET73	I	ITAKIM82B	I	11
ITAKIZAWA	I	11MMUI80	I	ITAKIM82C	I	11
IYAMAMOTO	I	51JIPNET73	I	IYAMAK75	I	51
IYAMAMOTO	I	51SCRTY	I	IYOKOM82A	I	11
IYAMAMOTO	I	515G79	I			
IYAMAMOTO	I	515G80	I			

DM1

DM3

図4.17 RWS内の結果

次に最終結果を得る為、DM3内の結果をDM1に送る。DM1内のDMQPは結合処理を行なう。処理を終了したら、DMQPはUSTに処理終了を通知する。

利用者は、処理終了の通知を受けた後、出力コマンドを使用して結果を出力する [図4.18]。

** TEMP			**
TEMPN	IREPNO	IPROJNO	I
ITAKIZAWA	ITAKIM82A	11DDBS80	I
ITAKIZAWA	ITAKIM82A	11JIPNET73	I
ITAKIZAWA	ITAKIM82A	11MMUI80	I
ITAKIZAWA	ITAKIM82B	11DDBS80	I
ITAKIZAWA	ITAKIM82B	11JIPNET73	I
ITAKIZAWA	ITAKIM82B	11MMUI80	I
ITAKIZAWA	ITAKIM82C	11DDBS80	I
ITAKIZAWA	ITAKIM82C	11JIPNET73	I
ITAKIZAWA	ITAKIM82C	11MMUI80	I
ITAKIZAWA	ITANAY81A	11DDBS80	I
ITAKIZAWA	ITANAY81A	11JIPNET73	I
ITAKIZAWA	ITANAY81A	11MMUI80	I
ITAKIZAWA	IYOKOM82A	11DDBS80	I
ITAKIZAWA	IYOKOM82A	11JIPNET73	I
ITAKIZAWA	IYOKOM82A	11MMUI80	I
IYAMAMOTO	IYAMAK75	11JIPNET73	I
IYAMAMOTO	IYAMAK75	11SCRTY	I
IYAMAMOTO	IYAMAK75	115G79	I
IYAMAMOTO	IYAMAK75	115G80	I

図4.18 最終結果

5. まとめと今後の課題

以上述べた様に、第Ⅰ部で述べた全体アーキテクチャ、第Ⅱ部のCODASYLスキーマから関係スキーマへの変換方法と、検索及び更新問合せの変換方法、及び第Ⅲ部で述べた分散問合せ処理方法とに基づいたJDDBSを実現することが出来た。JDDBSでは複数の既存CODASYL DBS上に関係モデルに基づいた1つの全体概念(GC)スキーマを定義し、これに対するRQL問合せを、分散処理し、問合せ変換処理して、解を関係として利用者に出力できる。更に、各CODASYL DBS上の関係インタフェースLDP(-V3)を通して、RQLによって非手続的な更新をデータベース(CODASYLデータベース)に対して行える。よって異種(CODASYL)DBSから成る統合的設計に基づいた異種分散型データベースシステムを実現出来たと考えている。現在、ホスト間の通信網が十分でない為に、ホスト間で通信を行なえず、M-170F内にAIM-DB/DCを用いて放送網をシミュレートしているが、DDBSレベルでの通信処理を実現出来たと考える。

今後の課題としては、以下がある。

- (i) ローカル網(LAN)の導入によって、実際の通信処理を行う。
- (ii) 大型機(e. g. M-170F)に加えて、パーソナルシステム上にDBMSを実現し、これを主要構成要素としていく。
- (iii) LDPの性能の改良。LDPは、現在毎回COBOL DMLを生成し、これをコンパイル・リンクし実行している。各DBS毎に、HIの生成時に実行形式のアクセスモジュールを生成し、これをコールする形にすることによって、コンパイル・リンク、及びデータベースのオープン・クローズ時間を減少させる。
- (iv) LDP上に、エディタとワードプロセッシングの一体化した利用者インタフェースを設ける。
- (v) 分散更新処理の実現。特に、B2Pプロトコルの実現。

参考文献

[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.

[ANDLS 82]

Andler, S., Ding, I., Eswaran, K., Hauser, C., Kim, W., Mehl, J., and Williams, R., "System D : A Distributed System for Availability," Proc. of the 8th VLDB, Mexico City, Sept. 1982, pp.33-44.

[BERNP 81]

Bernstein, P.A., and Goodman, N., "Concurrency Control in Distributed Database Systems," ACM Computing Surveys, Vol. 13, No.2, June, 1981, pp.185-221.

[BILLH 76]

Biller, H., Glatthaar, W., and Neuhold, E.J., "On the Semantics of Data Bases: the Semantics of Data Manipulation Languages," Proc. IFIP TC-2 Working Conf. on Modelling in Data Base Management Systems, Freudenstadt, Jan. 1976, pp.239-267.

[CODAS 73]

CODASYL, "CODASYL Data Description Language Journal of Development," June 1973. [NBS Handbook 113 (1974)].

[CODAS 78]

CODASYL DDL Committee, "Report of the CODASYL Data Description Language Committee," Information Systems, Vol. 3, No.4, 1978, pp.247-320.

[CODDE 70]

Codd, E.F., "A Relational Model of Data for Large Shared Data Bank," Communications of the ACM, Vol. 13, No. 6, June 1970, pp.337-387.

[DATEC 77]

Date, C. J., "An Introduction to Data Base System," (Second Edition), Addison-Wesley, June 1977, p.536.

[DAYAU 82]

Dayal, U. et al., "Query Optimization for CODASYL Database Systems," Proc. ACM SIGMOD, 1982, pp.138-150.

[HELDG 76]

Held, G.D., et al., "INGRES - A Relational Data Base System," AFIPS Conf. Proc., 1976, pp.409-416.

[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.

[KAMBY 82]

Kambayashi, Y., "Theory of Databases," Computer Science & Technologies, (Kitagawa, T. ed.), OHM & North-Holland, 1982, pp.206-226.

[LEBIJ 80]

Le Bihan, J., Esculier, C., Lann, G.L., Litwin, W., Gardarin, G., Sedilot, S., and Treille, L., "Sirius : A French Nationwide Project on Distributed Data Bases," Proc. of the 6th VLDB, Oct. 1980.

[OLLET 78]

Olle, T.W., "The CODASYL Approach to Data Base Management," John Willey & Sons, 1978.

[ROTHJ 80]

Rothnie, J.B., Bernstein, P.A., Fox, S., Goodman, N., Hammer, M., Landers, T.A., Reeve, C., Shipman, D.W., and Wong, E., "Introduction to a System for Distributed Databases (SDD-1)," ACM TODS, Vol. 5, No. 1, Mar. 1980, pp.1-17.

[SMITH 81]

Smith, J.M., Bernstein, P.A., Dayal, U., Goodman, N., Landers, T., Lin, K.W., and Wong, E., "Multibase - Integrating Heterogeneous Distributed Database Systems," AFIPS Conf. Proc., Vol. 50, Chicago, Ill., May, 1981, pp.487-499.

[TAKIM 78]

Takizawa, M., Hamanaka, E., and Ito, T., "Resource Integration and Data Sharing on Heterogeneous Resource Sharing System," Proc. ICCS'78, Kyoto, Japan, Sept. 1978, pp.253-258.

[TAKIM 79]

Takizawa, M. and Hamanaka, E., "The Four-Schema Concept as the Gross-Architecture of Distributed Databases and Heterogeneity Problems," Journal of Information Processing (JIP), IPSJ, Vol.2, Nov. 3, Nov. 1979, pp.134-142.

[TAKIM 80]

Takizawa, M., et al., "Query Translation in Distributed Databases," Proc. IFIP'80, Tokyo-Melbourne, Oct. 1980, pp.451-456.

[TAKIM 81]

Takizawa, M., "分散型データベースシステム JDDBS-II と通信処理", 電子通信学会 AL81-22, 1981.

[TAKIM 82a]

Takizawa, M., "Distribution Problems in Distributed Databases - Integration and Query Decomposition," JIP(IPSJ), Vol. 5, No. 3, 1982, pp.139-147.

[TAKIM 82b]

Takizawa, M., et al., "CODASYL データベースシステムに対する関係インタフェースシステム (LDP-V1.5) の設計と実現", IPSJ Vol, 23, No. 3, 1982, pp.665-675.

[TAKIM 82c]

Takizawa, M., "放送通信機を用いた分散型データベースシステムの通信処理実現について", IPSJ DBS 研資 31, 1982.

[TAKIM 82d]

Takizawa, M. and Noguchi, S., "CODASYL データベースシステムに対する
非手続的更新インタフェースの基本概念",
Jipdec TR 10/82, 1982.

[TAKIM 82e]

Takizawa, M. and Yokotsuka, M., "LDP-V1.5の総合評価",
データ・セマンティクスの理論と実際に関する研究
(数理解析研究所講究録 461)
京都大学数理解析研究所 1982、pp.127-150.

[TSICD 78]

Tsichritzis, D., and Klug, A., (eds.) "The ANSI/X3/SPARC
DBMS Framework," Report of the Study Group on Data Base
Management Systems. March. 1978 (also available from
Information Systems, Vol. 3, No. 3, pp.173-191, 1978.)

[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.

[WONGE 79]

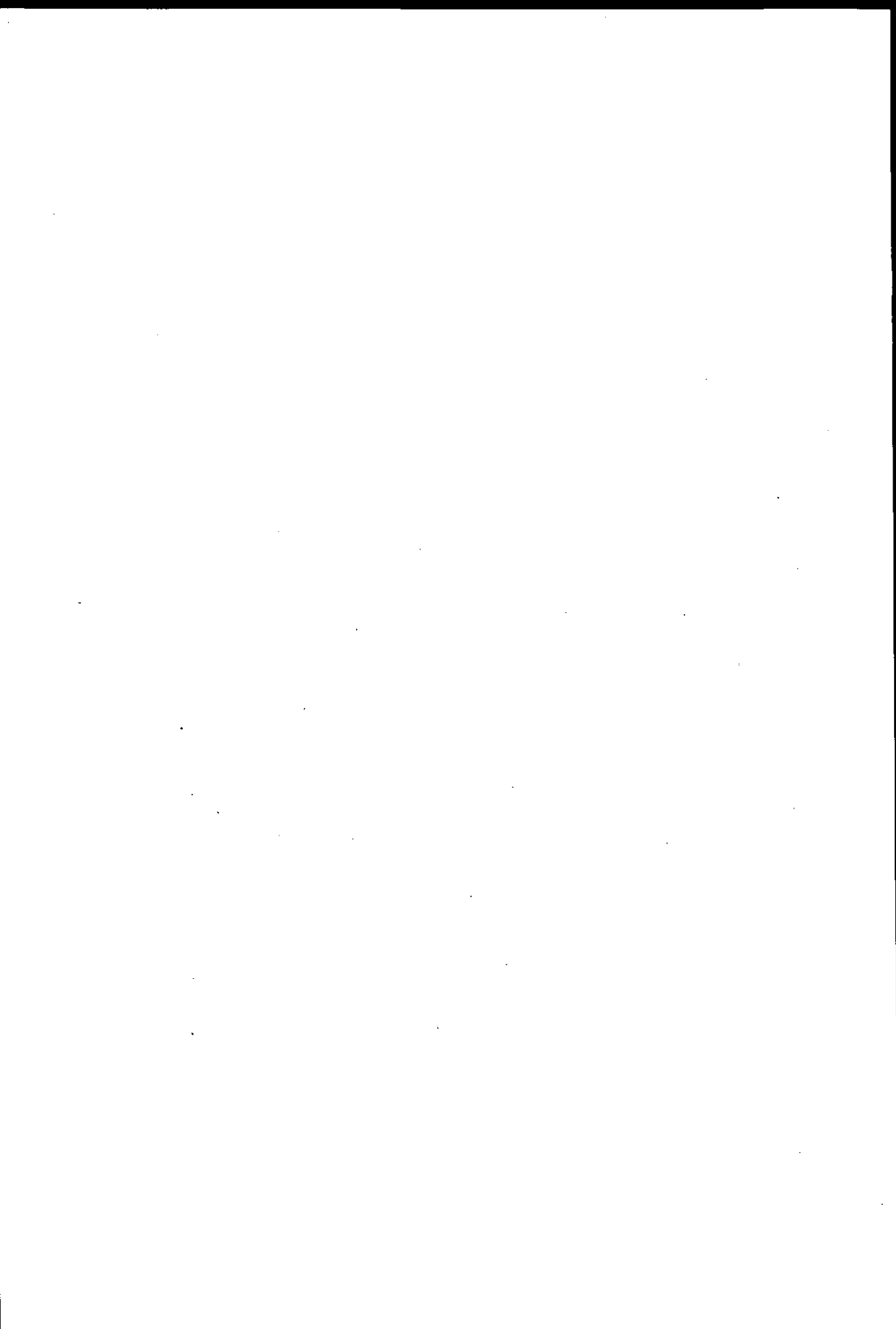
Wong, E., and Katz, R.H., "Logical Design and Schema Conversion
for Relational and DBTG Databases," Proc. of the International
Conf. on Entity-Relationship Approach to Systems Analysis and
Design, Los Angeles, CA., Dec. 1979, pp.311-321.

[YAMAH 81]

Yamazaki, H., "BANET," 沖電気, 1981.

[ZANIC 79a]

Zaniolo, C., "Design of Relational View Over Network Schemas,"
Proc. of the ACM SIGNOD International Conf. on Management of
Data, Boston, MA., May 1979, pp.179-190.



付 記

- 付 記 1 LDP コマンド
- 付 記 2 GDP コマンド
- 付 記 3 LDP 使用手引
- 付 記 4 DML I
- 付 記 5 集積関数処理の評価
- 付 記 6 発表論文リスト
- 付 記 7 英 文 資 料

付記1 LDP コマンド

ローカルデータベースプロセッサ (LDP) の利用者インタフェース (LDP コマンド) を示す。

1. L D P 狀態

LDPに対する利用者のアクセスは、`logon` コマンドでLDPセッションを開設する事によってなされる。図 1.1 は、LDP の状態を示している。表 1.1 には、各状態の意味を表している。

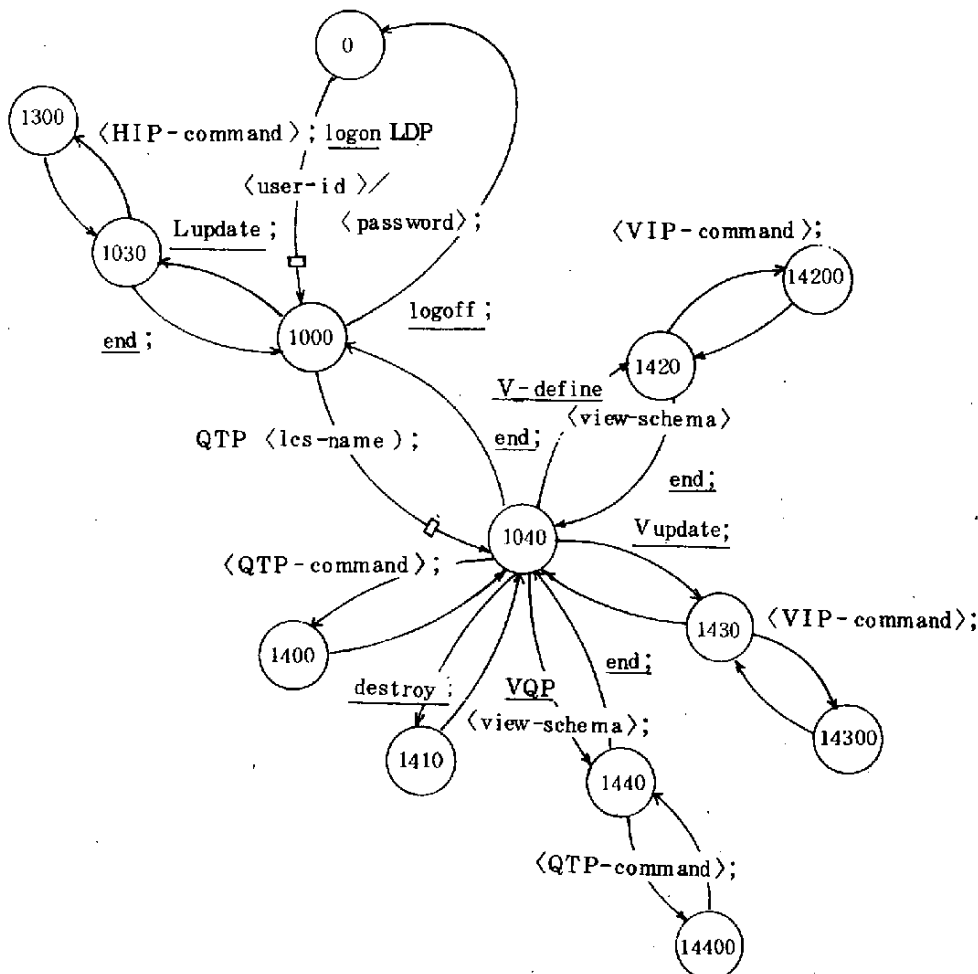


図 1.1 LDP の状態遷移図

表 1.1 L D P の 状 態

state 井	state の 意 味
0	initial state 全てのテーブルはclearされている
1 0 0 0	logged-on logonされLDPコマンドを待っている wait for LDP command
1 0 3 0	Lupdate 処理中 wait for HIP command
1 3 0 0	HIP command 処理中
1 0 4 0	QTP 処理中 wait for QTP command
1 4 0 0	QTP command 処理中 range, retrieve 等の処理中
1 4 2 0	VQP 処理中 wait for VQP command
1 4 2 0 0	VIP command 処理中 range, define 等の処理中
1 4 3 0	Vupdate 処理中 wait for VIP command
1 4 3 0 0	VIP command 処理中 range, define 等の処理中
1 4 4 0	VQP command 処理中 wait for QTP command
1 4 4 0 0	QTP command 処理中 range, retrieve 等の処理
1 4 1 0	destroy command 処理中

2. LDP コマンドクラス

LDP へのコマンドは、次の 4 つのクラスからなる。

- (1) 第 1 クラス state 0 から 1000 への遷移を行う。

logon LDP <user-id> / <password>;

 LDP session の開設を行う。

- (2) 第 2 クラス state 1000 から 10×× の state への遷移を行う。

 i) QTP <lcs-name>; QTP へのアクセス

 ii) Lupdate; <lcs> の定義

- (3) 第 3 クラス state 10×× から 1××× の state への遷移を行う。

 i) <HIP-command>; LCS の定義を行う

 ii) <QTP-command>; LCS 問合せの入力

 iii) Vdefine <view-schema-name>; 視野の定義を行う

 iv) Vupdate <view-schema-name>; 視野の更新を行う

 v) destroy <view-schema-name>; 視野の消去を行う

- (4) 第 4 クラス state 4 ケタ から 5 ケタ の state への遷移を行う。

 i) <VIP-command>; 視野の定義を行う

 ii) <QTP-command>; 視野問合せの入力

3. HIP コマンド

3.1 Lupdate コマンド

〔フォーマット〕 <lupdate-c> ::= Lupdate ;

〔機能〕 LCSの定義又は更新を行う為、HIPコマンドを使用することを宣言する。

〔解説〕 ローカル管理者(LA)だけが使用できるコマンドである。

```
Lupdate ;  
    { <HIP-command> }  
  
end ;
```

lupdate から end コマンドの間のHIPコマンドによって、LCS定義の修正を行う。

3.2 HIP コマンド

```
<HIP-コマンド> ::= <schema-c> | <rec-c> |  
    <set-c> | <DML-c> |  
    <rename-c> | <grant-c> |  
    <revoke-c> | <change>
```

HIPコマンドは、CODASYLスキーマ情報からLCSを生成しDML情報を生成するコマンドである。又、LCSの利用権の管理を行うコマンドである。HIPコマンドは、ローカル管理者(LA)のみが扱える。

A. SCHEMA コマンド

```
〔フォーマット〕 <schema-c> ::= schema <LCS-name>  
    { , <LIS-name> , <host-id> , <DB-name>  
        , <LA-name> } ;
```

〔機能〕 新しくLCS情報を定義する。このコマンド以下のRECコマンド、SETコマンドが、このLCS内のRECORD型、SET型として定義され、スキーマ変換されHIに格納される。

〔解説〕 新しく登録するLCS名<LCS-name>は、HI内でユニークでなければならぬ。LCS名、LIS名、ホスト名、DB名は16桁以内で、又LA名(ローカル管理者名)は20桁以内で定義される。

(例) LCS名(LCSPROJDB), LIS名(LISSUZUKI), Host-id(M170-F), DB名(PROJSUB4), LANAME(TAKIZAWA)に対するスキーマコマンドは、次の様である。

SCHEMA LCSPROJDB, LISSUZUKI, M170-F, PROJ SUB4, TAKIZAWA;

↑	↑	↑	↑	↑
LCS名	LIS名	host-id	DB名	LANAME

B. REC コマンド

```
[フォーマット] <rec-c> ::= rec : <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> ]
<format> ::= <type> <length>
<type> ::= C | D
<access-mode> ::= <calc-mode> / <dna-mode> /
                                [ / <nna-mode> ]
<calc-mode> ::= CALC | NCALC*
<dna-mode> ::= DNA | DA*
<nna-mode> ::= NNA | NA*
```

注) *は、default 値

〔機能〕 CODASYLスキーマのレコード型の情報から、LCS関係を生成する。

〔解説〕 レコード型内の項目は全て<field-list>内で定義する。リンクレコード型の場合、<rec-mode>としてLINKを指定する。CALC項目に対しては、<calc-mode>としてCALCを、DNA項目に対しては、<dna-mode>としてDNAを指定する。

<nna -mode>は、QTPコマンドでレコード型へ組を追加する時に、値の設定を省略できない項目に指定する。

エリア名、レコード型名、項目名は16桁以内で定義する。

〔例〕

area 名	record - type	冗長を許さない	append 時、省略できない項目
REC:RANGER	EMPLOYEE		
	(CNC /105 / / /DNA/ANA,		
	ENAME /C10,		
	EFNAME /C20,		
	ESEX /C01,		
	BIRTH-PLACE/C10,		
	BIRTH-YEAR /102,		
	BIRTH-MONTH/102,		
	ALMA-MATER /C30,		
	MAJOR /C30,		
	DEGREE /C01,		
	DEG-YEAR /102,		
	HIRED-YEAR /102,		
	RETIRED-YEAR/102,		
	DEPT /C20,		
	EPPOSITION /C20,		
	TEL /C12,		
	EXT /104		

C. SET コマンド

〔フォーマット〕 <set-c> ::= set <set-type> <owner-description>
 <member-description> [<structural constraint>] ;

<owner-description> ::= OWNER <record-type>
 <member-description> ::= MEMBER <record-type>
 [/ <access-mode>]

<access-mode> ::= <dna-att> / <sort-att> [/ <link-
 -mode>] [/ <insertion-mode> /
 <retention-mode>]

<dna-att> ::= <attribute-name> | NIL
 <sort-att> ::= <attribute-name> | NIL
 <insertion-mode> ::= AUTOMATIC* | MANUAL
 <retention-mode> ::= FIXED | OPTIONAL |
 MANDATORY*

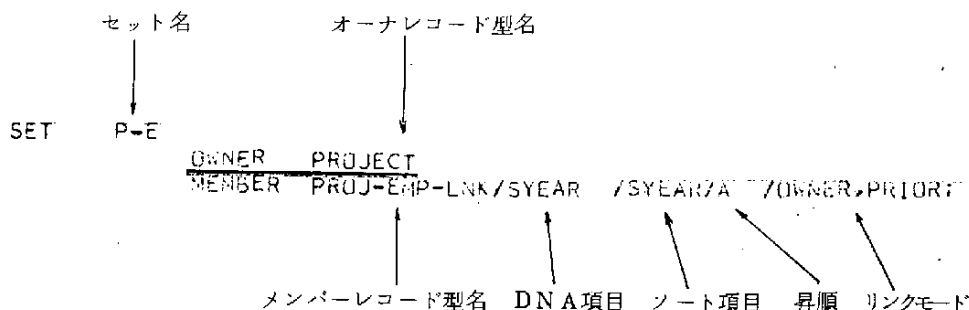
<link-mode> ::= NEXT | OWNER | PRIOR
 <structural constraint> ::= STRUCTURE (<att-
 name-of-o-rec> ,
 <att-name-of-m-rec>)

〔機能〕 CODASYLスキーマのセット型の情報から、LCS関係を生成する。

〔解説〕 セット型でDNA項目がある場合は、<dna-att>で指定する。SORT項目がある場合は、<sort-att>で指定する。メンバシップクラスは、各々、<intension-mode>と<retention-mode>で指定する。

セット型名、オーナーレコード型名、メンバレコード型名は16桁以内で定義する。DNA項目、ソート項目は各々、メンバレコード型内に定義される項目でなければならない。

〔例〕



D. DML コマンド

〔フォーマット〕 <DML-c> ::= <DML-block-def> { <DML-block-def> }
<DML-block-def> ::= <block-no> // <DML-statement> { // <DM-statement> };
<block-no> ::= P <number>

〔機能〕 COBOL DMLパターンの定義をHI情報に生成する。

〔解説〕 DMLパターンの入力情報は、非常に多くなる(ステートメント数約1000行)ので通常バッチで処理する。

E. RENAME コマンド

〔フォーマット〕 <rename-c> ::= rename <rel-name1> by <rel-name2>
| rename <att-name1> in <rel-name1>
| by <att-name2>;

〔機能〕 LCS内の関係名、あるいは属性名を変更する。

〔解説〕 変更される関係名、あるいは属性名はLCS内に存在しなければならない。

例えば、従業員 (EMPLOYEE) の属性名 (@EMPLOYEE) を、属性名 (@E) に変更する場合、RENAME コマンドは次の様である。

RENAME EMPLOYEE OF EMPLOYEE BY DE:

F. GRANT コマンド

```

〔フォーマット〕  <grant-c> ::= grant <les-name> { , <access-list> }
                                     to <user-id> { , <user-id> } ;
<access-list> ::= ALL * | Retrieve | Append |
                                     Replace | Delete

```

〔機能〕 利用者に、LCS<les-name>の利用権を与える。GRANT コマンドは、LCS 管理者だけが使用できる。

〔解説〕 LDPを利用する利用者に対して、LDPが提供するデータベース（HIPによってLCSが登録済みのもの）の利用権を与える。＜user-id＞は、LCS管理者によって登録されていないといけない。

<access-list>は、利用者に与えられる利用権に対して、検索・更新（追加、変更、削除）のうちどれを与えるかを指定する。省略した場合は、規定値としてALLがとられ、検索及び更新権が利用者に与えられる。

〔例〕 DCDBS (LCS名="LCSDCDBS")の利用権を利用者 (user-id="KOSEKI")に与える場合、GRANTコマンドは次の様である。

GRANT LCSDCDBS TO KOSEKI;

G. REVOKEコマンド

```
[フォーマット] <revoke-c>::=revoke<user-id>{,<user-id>}
                                from<les-name>;
```

〔機能〕 利用者(<user-id>)から、データベース<ldb-name>の利用権を取り上げる。

〔解説〕 利用者<user-id>に対して、LDPがサポートするデータベース<ldb-name>の利用権を取り上げる。

〔例〕 利用者 (user-id = "KOSEKI") から、DCDBS (LCS-name = "LCSDCDBS") の利用権を取り上げる場合、REVOKE コマンドは次の様である。

REVOKE KOSEKI FROM LCSDCDBS;

H. CHANGE コマンド

[フォーマット] <change - c> ::= change <useuid> / <password>
to <password>;

[機能] 利用者のパスワードの変更を行う。

[例] 利用者 (利用者識別名 = "YOKOM"、パスワード = "YOKOM") のパスワードを "MINOR" に変更する場合、CHANGE コマンドは次の様である。

CHANGE YOKOM/YOKOM TO MINOR;

4. QTPインタフェース

QTPインタフェースは、利用者にLCSへの問合せ、視野の定義、更新、削除、及び視野への問合せを行うQTPコマンドを提供する。

QTPコマンドは、次の様に使用される。

```
logon LDP <user-id>/<password>;
#9999 QTP <lcs-name>;
```

利用者が参照するLCS<lcs-name>を宣言する。これにより、QTPセッションが開設される。QTPセッション内では、次の様な処理を実行することができる。

(1) LCSへの問合せ

```
#9999 <QTP-command>;
LCSへの問合せを定義し、実行させる。
```

(2) 視野の定義

```
#9999 Vdefine <view-schema-name>;
{ V9999 <VIP-command>; }
V9999 end;
```

視野<view-schema-name>を新たに定義することを宣言する。

(3) 視野の更新

```
#9999 Vupdate <view-schema-name>;
{ V9999 <VIP-command>; }
V9999 end;
```

視野スキーマ<view-schema-name>を更新するため視野の定義式をLDPへロードする。

(4) 視野の削除

```
#9999 destroy <view-schema-name>;
視野スキーマをHIから削除する。
```

(5) 視野への問合せ

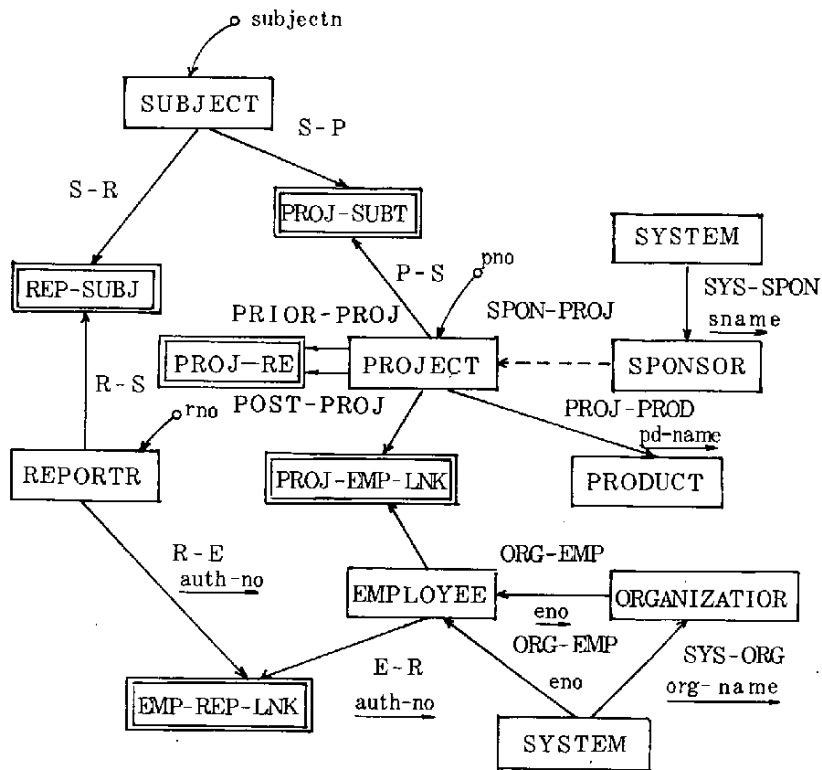
```
#9999 VQP <view-schema-name>;
{ V9999 <QTP-command>; }
V9999 end;
```

参照する視野<view-schema-name>の情報をLDPにロードする。

(6) QTPセッションを終了する。

```
#9999 end;
QTPセッションを終了する。
```

QTPコマンドの例で使用するPRDBSのLIS(CODASYLスキーマ)を図1.2に示す。



: レコード型 $\longrightarrow$: セット型 (M/A)
 : リンクレコード型 $\dashrightarrow$: セット型 (O/M)

$\circ \longrightarrow$: CALC 項目

$\xrightarrow{\alpha}$: Sort key 項目

図 1.2 PRDBS の LIS

HIPによって変換されたPRDBSのLCSは以下の様である。このLCSに対するQTPコマンドの例を示す。

a) E型関係

SUBJECT (@J, subjectn)

SPONSOR (@S, sname, stype)

PROJECT (@P, pno, pname, proj-type, status-f,
year, eyear, budget)

PRODUCT (@D, pd-no, pd-name, ptype, cost, year, language, psize)

EMPLOYEE (@E, eno, ename, efname, esex, birth-place, birth-year,
birth-month, alma-mater, major, degree, deg-year,

hired-year,retired-year,dept,position,tel,ext)

REPORTR (@R,rno,title,source-n,val,numb,month,year,from-page,
to-page,total-page,no-of-author,language,written-year,
cost,rtype,rstate)

ORGANIZATIOR (@O,org-name,org-type)

b) G型関係

PROJ-EMP-LNK (@P,@E,smmonth,eyear,emonth,pposition,role,
percentage)

PROJ-SUBJ (@P,@J)

REP-SUBJ (@R,@J)

EMP-REP-LNK (@E,@R,auth-no)

PROJ-RE (@P,prior-pr-p)

c) B型(T)関係

ORG-EMP (@O,@E)

PROJ-PROD (@P,@D)

d) B型(P)関係

SPON-PROJ (@S,@P)

4.1 QTPコマンド

(フォーマット) QTP<lcs-name>;

(機能) LCS<lcs-name>への副セッションを開設する。

[解説] LCS<lcs-name>への検索、更新、視野の定義、更新、視野への検索、更新を行う
為には、まずQTPコマンドで<lcs-name>をもつデータベースへの副セッションを開設せね
ばならない。QTPコマンドを利用出来る利用者は、あらかじめローカル管理者から利用権を得
ている必要がある。

```
QTP <lcs-name>;  
    {<QTP-mc>;}  
end;
```

4.2 QTP主コマンド

```
<QTP-mc>::=<QTP-c>|<VQP-c>  
          <destroy-c>|<Vdefine-c>  
          <Vupdate-c>|<end-c>
```

QTP主コマンドは、LCS又は視野に対するアクセスする為、あるいは視野を定義する為のコマンドである。QTP主コマンドを用いれる利用者は、ローカル管理者によってあらかじめ利用権が与えられていなければならない。

4.3 QTPコマンド

```
<QTP-コマンド> ::= <range-c> | <retrieve-c> |
                    <append-c> | <delete-c> |
                    <replace-c> | <go-c> |
                    <outp-c> | <modec-c> |
                    <erase-c> | <status-c> |
                    <display-c> | <edit-c> |
                    <dump-c> | <help-c> |
                    <DFA-C>
```

QTPコマンドは、LCS内の関係に対する検索、あるいは更新問合せを作成し、実行させるコマンドである。

A. RANGEコマンド

```
[フォーマット] <range-c> ::= range <range-def> { <range-def> } ;
                <range-def> ::= ( <var-list> , <lrel-name> )
                <var-list> ::= <var> | ( <var> { , <var> } )
```

[機能] 問合せで参照されるLCS関係<lrel-name>に対する組変数<var>を宣言する。

[解説] 問合せで参照するLCS関係<lrel-name>に対して組変数<var>を宣言する。宣言された組変数は、LDPセッションが終了するまで(LOGOFFコマンド)有効となる。異なる関係に、同一の組変数名を宣言した場合、既に宣言されていた組変数を新しく宣言されたもので置換するか選択することができる。組変数名は、4桁以内で、頭文字は英文字でなまればならない。

組変数は、最大32個まで宣言することができる。

B. RETRIEVEコマンド

```
[フォーマット] <retieve-c> ::= retrieve into <rel-name>
                    ( target-list ) [ where qual ] ;
```

[機能] 条件式(qual)を満足する全組を検索し、結果を結果作業領域(RWS)に格納する。

〔解説〕 検索の結果は、指定された関係名でRWSに格納される。結果関係は、利用者毎に、ユニークな関係名で管理される。従って、同一関係名を持つ結果がRWSに存在しているか実行時(GOコマンド)にチェックされる。同一関係名が存在している時、問合せは実行されない。関係名を変更するか、RWSの結果を削除する(ERASEコマンド)。

結果は、LDPセッション終了(ENDコマンド)時まで保存され、終了時に削除される。結果を保存する為には、結果のモードを永久モードに変更する(MODECコマンド参照)。

結果関係名は、16桁以内で、頭1文字は英文字でなければならない。

〔例〕 プロジェクト番号が"MMUI80"であるプロジェクトに属する従業員の名前、出身地、プロジェクト番号を求める場合、RETRIEVEコマンドは次の様である。

```
RANGE (E,EMPLOYEE);  
RANGE (P,PROJECT);  
RANGE (PE,PROJ-EMP-LNK);  
RETRIEVE INTO R (P.PNO, E.ENAME, E.BIRTH-PLACE)  
WHERE  
  E.OE = PE.OE AND PE.OP = P.OP AND  
  P.PNO = "MMUI80";  
GO;
```

(1) target-list

〔フォーマット〕 <target-list> ::= [<name>=] <var> * <att-name>
[, <name>=<aexp>] |
<att-name>=<constant>
[, <att-name>=<var>* <att-name>]

〔機能〕 結果として導出される関係の属性を定義する。

〔解説〕 検索問合せと更新問合せとでは、target-listのフォーマットが異なる。

1) 検索問合せ

<target-list> ::= [<name>=] <var> , <att-name>
[, <name>=<aexp>]

検索の結果関係の属性は、組変数<var>とその組変数が参照する関係の属性<att-name>によって定義される。この時、結果関係の属性として、新たに属性名<name>を与えることができる。<name>=を省略した場合、<att-name>がそのまま結果関係の属性名となる。結果関係内に同一属性名が発生するときは、属性名に番号を付加して区別する。

<name>=<aexp>の場合には、算術式<aexp>の値が属性<name>の値となる。

ii) 更新問合せ

```

<target-list> ::= <att-name> = <constant> |
                <att-name> = <aexp> |
                <att-name> = <var> . <att-name>

```

更新問合せに対しては、更新する関係の属性<att-name>の値を設定する。定数<constant>、算術式<aexp>あるいは、組変数<var>とその組変数が参照する関係の属性によって定義する。但し、算術式は、関係の属性のタイプが数字タイプだけに設定できる。

(2) qual

```

〔フォーマット〕 <qual> ::= <c-pred> | ( <qual> ) |
                  NOT <qual> | <qual> AND <qual> |
                  <qual> OR <qual>
<c-pred> ::= <aexp> <cop> <aexp>
<aexp> ::= <constant> |
           <var> . <att-name> |
           ( <aexp> ) |
           <aexp> <aop> <aexp> |
           - <aexp> |
           <g-function>
<cop> ::= LT | LE | EQ | GE | GT | NE |
          <| <= | = | >= | > | !=
<aop> ::= + | - | * | / | **
<g-function> ::= aggregate 関数
                  ( aggregate 関数を参照 )

```

〔機能〕 問合せが満たす条件式を定義する。

〔解説〕 条件式で参照される組変数は、新じめRANGEコマンドで宣言しておく。

例題を通じて、条件式の定義方法を述べる。

i) 従業員 (EMPLOYEE, 組変数は e) に対して、従業員番号が10番以内という条件は、次の様である。

```
e.eno LE 10
```

ii) 従業員番号が、10～30までという条件は、次の様である。

```
e.eno GE 10 and e.eno LE 30
```

iii) 従業員番号が10以下、又は30以上という条件は次の様である。

e.eno LE 10 or e.eno GE 30

iv) 上の条件に出身地が東京である条件を加えた条件は、次の様である。

(e.eno LE 10 or e.eno GE 30) and e.birth-place
= "TOKYO"

(属性が文字タイプの場合、文字定数はダブルクォーテーションで囲う。)

v) 従業員とレポートの関係を条件とする場合、条件は関係の主キーを用いて、次の様である。

e.@E=er.@E and er.@R=r.@R
(rはREPORTR, erはEMP-REP-LNKの組変数)

[制限]

qualは、次の様に積正規化されているとする。

qual ::= C_i and and C_n

C_i ::= rei(x) | jci(x,y)

rei(x) ::= rpi_i(x) or or rpi_i(x)
(i = 1, ..., n)

jci(x,y) ::= pjp(x,y) | njpi_i(x,y) or or njpi_i(x,y)
(i = 1, ..., 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。

rp(x)は制限述語、pjp(x,y)は主結合述語、njp(x,y)は非主結合述語を表す。

(3) 集積 (aggregate) 関数 (GF)

[フォーマット] GF ::= <s-g-func> | <g-g-func>

i) simple aggregate (SA)

<s-g-func> ::= <g-func-name> (<exp> [, <u-att> { , <u-att> }])
(where <qual>))

ii) group-by aggregate (GA)

<g-g-func> ::= <g-func-name> (<exp> [, <u-att> { , <u-att> }])
by <vatt> { by <vatt> }
(where <qual>))

<g-func-name> ::= max | min | any | count | aver | sum

[機能]

i) simple aggregate

条件式<qual>を満たす組の集合に対して、aggregate処理を行いスカラー値を与える。

ii) group-by aggregate

by <vatt>毎に、条件式<qual>を満たす組の集合が求められ、aggregate処理される。

〔解説〕 aggregate関数に対して、次の制限がある。

- (1) aggregate関数の参照する変数は、全て主等価結合 (primary equi-join) によって連結される。
- (2) simple aggregate関数は、その内部にgroup-by aggregate関数を含んではならない。
- (3) group-by aggregate関数は、その内部にgroup-by aggregate関数を含んではならない。

〔simple集積関数の例〕

- ・従業員番号 (eno) が10以下の従業員の平均年令を求める場合、問合せは次の様である。

```
RANGE (E,EMPLOYEE);  
RETRIEVE INTO R5 (AGE= AVER (E.BIRTH-YEAR, E.@EMPLOYEE  
WHERE E.ENO <= 10));
```

- ・従業員の平均年令よりも大きい従業員の属しているプロジェクトの番号 (pno)、従業員の生年 (birth-year)、平均年令を求める場合、問合せは次の様である。

```
RANGE (E,EMPLOYEE);  
RANGE (P,PROJECT);  
RANGE (PE, PROJ-EMP-LNK);  
RETRIEVE INTO SG003 (P.PNO, E.BIRTH-YEAR,  
A1 = AVER (E.BIRTH-YEAR, E.@E))  
WHERE  
E.BIRTH-YEAR >= AVER (E.BIRTH-YEAR, E.@E) AND  
E.@E = PE.@E AND PE.@P = P.@P;
```

〔group-by集積関数の例〕

- ・プロジェクト毎に、そのプロジェクトに属している平均年令とプロジェクト番号を求める場合、問合せは次の様である。

```
RANGE (P, PROJECT);  
RANGE (E, EMPLOYEE);  
RANGE (PE, PROJ-EMP-LNK);  
RETRIEVE INTO GG001 (P.PNO, A1=AVER(A2 = E.BIRTH-YEAR, E.@E  
BY P.@P WHERE E.@E=PE.@E AND PE.@P=P.@P));
```

・著者数が2以上の著者(AUTHORR)の中で、最も価格の高い本(DOCUMENT)を書いている著者、著書(title)、コストを求める場合、問合せは次の様である。

```
RANGE (D,DOCUMENT) (DA,DOC-AUTH) (A,AUTHORR);  
RETRIEVE INTO ROI (LN=A.LAST-NAME,NQ=D.JND,TL=D.TITLE,  
COST=D.COST)  
WHERE  
D.COST=MAX(D.COST) AND  
COUNT (D.JND BY A.LAST-NAME BY A.FIRST-NAME  
WHERE D.@D = DA.@D AND DA.@A = A.@A ) GE 2;
```

C. APPENDコマンド

[フォーマット] <append-c> ::= append to <rel-name>
(<target-list>) [where <qual>] ;

[機能] 条件式<qual>を満たす組の集合を関係<rel-name>に追加する。

[解説] 関係<rel-name>は、QTPコマンドで指定したLCS<lcs-name>内に存在する関係でなければならない。<target-list>は、関係に追加される属性の値を定める式を記述する。結果属性として、<target-list>に表われないnull値が許される属性に対して、文字タイプはブランク、数字タイプはゼロを値として与える。null値の許されない属性が<target-list>から抜けている時は、エラーとなる。

追加する関係がE型関係、B型(T,P)関係、G型関係によって次の様な制限がある。

(1) E型関係(レコード型に対応)の組の追加

ここでは、次の条件が満足されなければならない。

- i) <target-list>内の結果属性に関係の主キーがあってはならない。例えば、PRDBSのEMPLOYEEに組を追加する場合、結果属性に主キー@Eがあってはならない。
- ii) <target-list>内の結果属性は、関係内のnull値が許されない属性については全て与えられなければならない。

例えば、PRDBSのEMPLOYEEの場合、null値が許されない属性enoは与えられなければならない。

- iii) E型関係の主キーがB型(T)関係の主キーになっている場合(MANDATORY/AUTOMATICセット型の子レコード型に対応)、B型(T)関係で関連づけられる他方のE型関係の組は<qual>によって、追加される組に対して唯一つに決まらなければならない。

例えば、PRDBSのE型関係EMPLOYEEの場合、B型(T)関係ORG-EMPの主キーとなっているので、E型関係ORGANIZATORの組を<qual>で規定する。

(2) B型関係(セット型に対応) $B(@E_1, @E_2)$ への組の追加

B型関係に組を追加する場合、次の条件を満足しなければならない。

i) $\langle \text{target-list} \rangle ::= @E_1 = e_1 \cdot @E_2 = e_2 \cdot @E_2$

ii) B型関係はT型(MANDATORY/AUTOMATICセット型)であってはならない。

iii) $\langle \text{qual} \rangle$ は、関連づけられるE型関係の組(e_1, e_2)を決めるものである。

(3) G型関係(リンクレコード型に対応) $G(@E_1 \dots, @E_n, b_1 \dots, b_m)$ への組の追加

G型関係に組を追加する場合、次の条件を満足しなければならない。

i) $\langle \text{target-list} \rangle ::= @E_1 = e_1 \cdot @E_2, \dots, @E_n = e_n \cdot @E_n \quad [, b_1 = \text{exp}_1]$
 $\dots [, b_m = \text{exp}_m]$

ii) $\langle \text{target-list} \rangle$ 内には、 $\langle \text{rel-name} \rangle$ 内のnull値が許されない属性は全て含まなければならない。

iii) $\langle \text{rel-name} \rangle$ が関連づけるE型関係の組($e_1, \dots, e_n$)は、全て $\langle \text{qual} \rangle$ によって決められなければならない。

[例]

(i) PRDBSでEMPLOYEEの組を追加する場合、APPEND文は次の様である。

```
RANGE (O.ORGANIZATION);
APPEND TO EMPLOYEE
(ENO = 44, ENAME = "MUKAIYAMA", EFNAME = "HIROSHI")
WHERE
  O.ORG-NAME = "JIPDEC";
```

LCS

EMPLOYEE (E型関係)

ORG-EMP (B型(T)関係)

eno	ename
	TAKIZAWA
.	.
.	.
.	.

@O	@E
(JIPDEC)	(TAKIZAWA)
.	.
.	.
.	.

追加する組

44	MUKAIYAMA
----	-----------

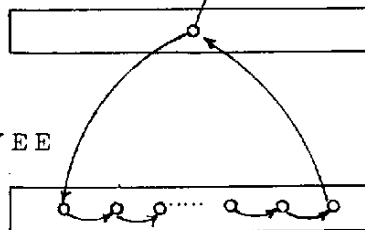
(JIPDEC)	(MUKAIYAMA)
----------	-------------

(JIPDEC) = "JIPDEC" の @O 値 (MUKAIYAMA) = "MUKAIYAMA" の @E 値

L I S

ORGANIZATOR org-name = "JIPDEC"

EMPLOYEE



ename = "MUKAIYAMA" の追加

(ii) G型関係EMP-REP-LNKへの組の追加は、次の様に記述される。

```
RANGE (R,REPORTR);
RANGE (E,EMPLOYEE);
APPEND TO EMP-REP-LNK
( @E=E.@E , @R=R.@R, AUTH-NO=1)
WHERE
  E.ENAME = "TAKIZAWA" AND
  R.RNO = "TAKIM81H";
```

L C S

EMP-REP-LNK (G型関係)

@E	@R
(TAKIZAWA)	(TAKIM81A)
.	.
.	.
.	.

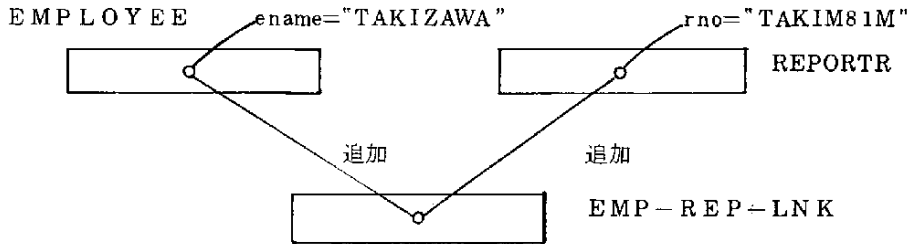
追加する組

(TAKIZAWA)	(TAKIM81H)
------------	------------

追加

@E	@R
(TAKIZAWA)	(TAKIM81A)
.	.
.	.
.	.
(TAKIZAWA)	(TAKIM81H)

L I S



- (iii) 主題 (subjectn) が "GLOBAL CONCEPTUAL SCHEMA" であるレポート (REPORTR) に、主題 (subjectn) として "GCS" を追加する APPEND コマンドは、次の様である。

```

RANGE (J1,J), SUBJECT);
RANGE (R,REPORTR);
RANGE (RJ,REP-SUBJ);
APPEND TO REP-SUBJ (@R=R.@R, @J=J.@J)
WHERE
  J.SUBJECTN="GCS" AND
  J1.SUBJECTN="GLOBAL CONCEPTUAL SCHEMA" AND
  J1.@J = RJ.@J AND RJ.@R = R.@R;
  
```

L C S

REPORTR (E型関係) REP-SUBJ (G型関係) SUBJECT (E型関係)

@R	
(R1)	
(R2)	
(R3)	
(R4)	

@R	@J
(R1)	(J1)
(R1)	(J3)
(R2)	(J3)
(R3)	(J1)
(R3)	(J2)
(R3)	(J3)
(R4)	(J3)

@J	
(J1)	GLOBAL.....
(J2)	
(J3)	
(J4)	GCS



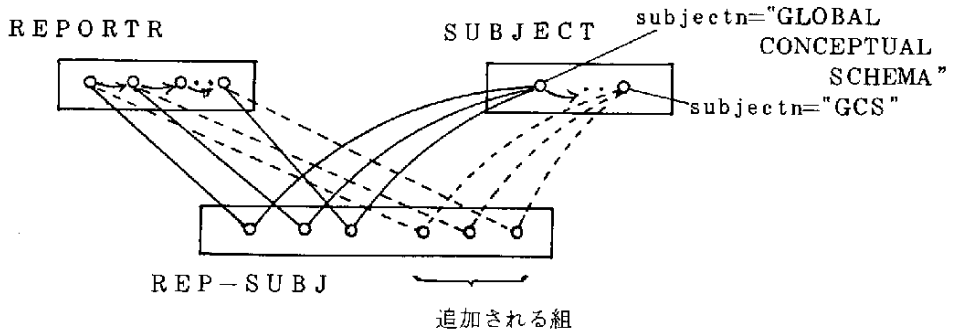
append コマンドの実行結果

@R	@J
(R1)	(J1)
(R1)	(J3)
(R1)	(J4)
(R2)	(J3)
(R3)	(J1)
(R3)	(J2)
(R3)	(J3)
(R3)	(J4)
(R4)	(J3)

← 追加

← 追加

L I S



D. REPLACEコマンド

〔フォーマット〕 `<replace-c> ::= replace <var>`
`( <target-list> ) ( where <qual> ) ;`

〔機能〕 条件式<qual>を満たす組の結果属性値を、<target-list>で記述された値で置換する。

〔解説〕 組変数<var>は、変更する関係に対して宣言しておく。<target-list>内に表われる属性だけが置換される。

変更される関係、E型関係、B型(T, P)関係、G型関係によって次の制限がある。

(1) E型関係の組の置換

ここでは、次の条件が満足されなければならない。

- i) E型関係の組の内、変更したい属性とその値を<target-list>に記述する。
- ii) L I SのCALC項目に対応した属性の置換は出来ない。
- iii) 主キーの置換は出来ない。

(2) G型関係の組の置換

条件はE型関係の組の置換の場合と同じである。(但し、主キーの変更は行えるとする。)

(3) B型関係B(@E₁, @E₂)の組の置換

関係間の関係性を表わす組を変更する。ここでは、次の条件が満足されなければならない。

- i) 主キー値の置換は出来ない。
- ii) `<target-list> ::= @E1 = e1 . . @E1`
`e1` は<qual>で規定する。

〔例〕 (i) 従業員名が" MORO "である組の出身地を" TOKYO "に変更する場合、REPLACE文は次の様である。

```

REPLACE E (BIRTH-PLACE = "TOKYO" )
WHERE
  E.ENAME = "MORO";
GO;

```

LCS

EMPLOYEE (E型関係)

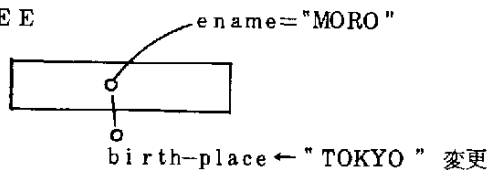
ename	birth	
TAKIZAWA	TOKYO	
MORO		

変更
→

ename	birth	
TAKIZAWA	TOKYO	
MORO	TOKYO	

LIS

EMPLOYEE



- (ii) プロジェクト "C858" と主題 (subjectn) "COMPUTER NETWORKS" との関係プロジェクト "C858" と主題 (subjectn) "DATABASE SYSTEMS" に変更する場合、replace 文は次の様である。

```

RANGE ((J,J1),SUBJECT);
RANGE ((P,P1),PROJECT);
RANGE (PJ,PROJ-SUBJ);
REPLACE PJ ( @P=P.@P, @J=J.@J )
WHERE
  P.PNO = "C858" AND J.SUBJECTN = "DATABASE SYSTEMS" AND
  P1.PNO = "C858" AND J1.SUBJECTN = "COMPUTER NETWORKS" AND
  P1.@P = PJ.@P AND J1.@J = PJ.@J;

```

L I S

PROJECT (E型関係) SUBJECT (E型関係) PROJ-SUBJ (G型関係)

pno
MMUI80
.
.
.
C858

subjectn
COMPUTER
.
.
.
DATABASE...

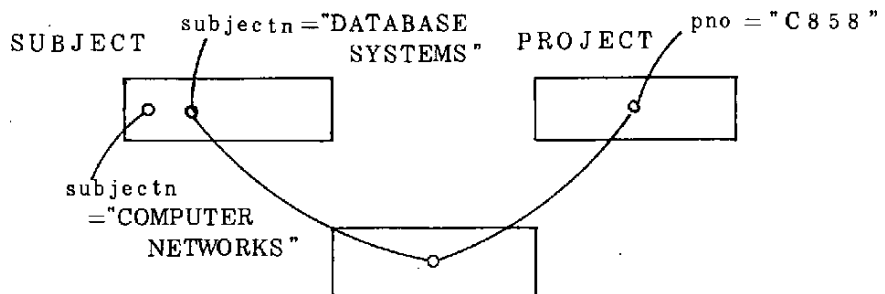
@P	@J
(MMUI80)	(COMPUTER...)
(MMUI80)	(DATABASE...)
.	.
.	.
.	.
(C858)	(COMPUTER...)

↓ 変更

@P	@J
(MMUI80)	(COMPUTER...)
(MMUI80)	(DATABASE...)
.	.
.	.
.	.
(C858)	(DATABASE...)

変更→

L I S



(iv) EMPLOYEE (ename="xx") の所属する組織 (ORGANIZATION) を "TOKYO UNIVERSITY" に変更する場合、REPLACE文は次の様である。

```

RANGE (E,EMPLOYEE);
RANGE (O,ORGANIZATION);
RANGE (OE,ORG-EMP);
REPLACE OE ( 20=0.20 )
WHERE
  E.ENAME = "XX" AND
  E.OE = OE.OE AND O.ORG-NAME = "TOKYO UNIVERSITY";

```

LCS

ORG-EMP (B型(T)関係)

@E	@O
(TAKIZAWA)	(JIPDEC)
⋮	⋮
(× ×)	(JIPDEC)

ORG-EMP

@E	@O
(TAKIZAWA)	(JIPDEC)
(× ×)	(TOKYO…)

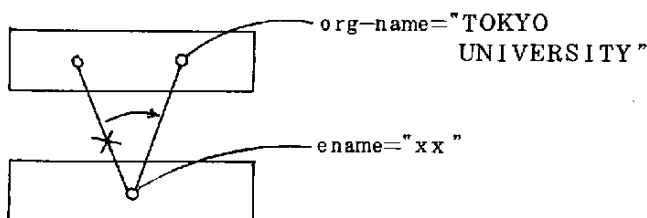
変更

LIS

ORGANIZATION

org-name="JIPDEC"

EMPLOYEE



E. DELETE コマンド

〔フォーマット〕 <delete-c> ::= delete <var> [where <qual>] ;

〔機能〕 条件式<qual>を満たす組の集合を、これらが属する関係(組変数で参照される関係)から削除する。

〔解説〕 組変数<var>は、range コマンドで宣言されていないといけない。条件式が無い場合、その関係内の全組を削除する。

削除する関係の型によって、次の様に更新が伝播する。

(1) E型関係の組の削除

- i) E型関係がG型関係によって関連づけられている時、対応するG型関係の組を削除する。
- ii) E型関係がB型(P)関係によって関連づけられている時、対応するB型(P)関係の組を削除する。
- iii) E型関係がB型(T)関係によって関連づけられている時、対応するB型(T)関係の組を削除する。E型関係の主キーがB型(T)関係の主キーでない場合(M/Aセット型

の親レコード型に対応する場合)、B型(T)関係の主キーに対応するE型関係の組を削除する。この削除は伝播する。

(2) G型関係の組の削除

G型関係の組だけが削除される。

(3) B型(P)関係の組の削除

B型(P)関係の組だけが削除される。

(4) B型(T)関係の組の削除

B型(T)関係の組を削除する場合、B型(T)関係の主キーに対応するE型関係の組も削除する。この削除は伝播する。

〔例〕

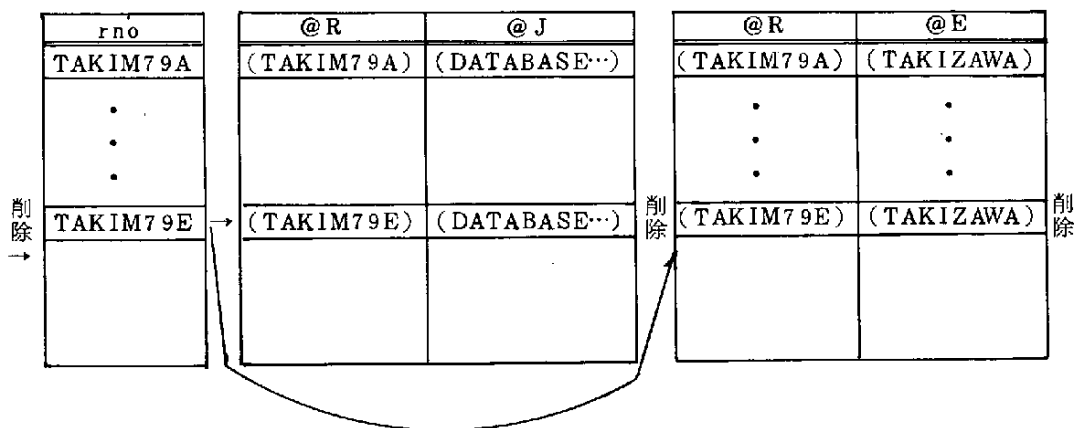
(1) レポート番号(rno)が" TAKIM79E "である組をレコード型REPORTRから削除する場合、DELETE文は次の様である。

```
RANGE (R,REPORTR);
DELETE R
WHERE R.RNO = "TAKIM79E";
GO;
```

LCS

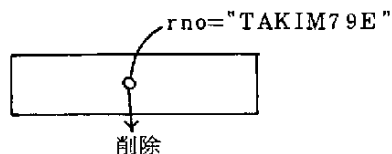
REPORTR (E型関係) REP-SUBJ (G型関係)

EMP-REP-LNK (G型関係)



LIS

REPORTR



- (iii) 従業員番号 (eno) が 20 の従業員の組を EMPLOYEE から削除する場合、DELETE 文は次の様である。

```
RANGE (E,EMPLOYEE);  
DELETE E WHERE E.ENO = 20;
```

LCS

EMPLOYEE (E型関係)

eno	ename
1	TAKIZAWA
20	xx

ORG-EMP (B型(T)関係)

@O	@E
(JIPDEC)	(TAKIZAWA)
.	.
.	.
(JIPDEC)	(xx)

EMP-REP-LNK (G型関係)

@E	@R
(TAKIZAWA)	(TAKIM79A)
.	.
.	.
(xx)	(PNO)

PROJ-EMP-LNK (G型関係)

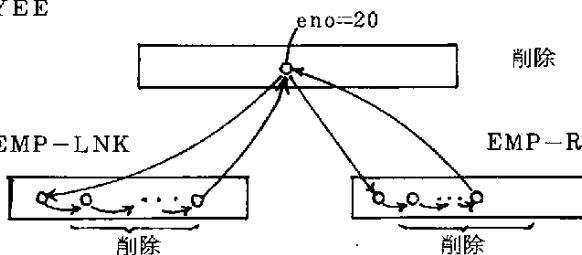
@E	@P
(TAKIZAWA)	(MMUI80)
.	.
.	.
(xx)	(PNO)

LIS

EMPLOYEE

PROJ-EMP-LNK

EMP-REP-LNK



(iii) プロジェクト番号が "C858" で、主題が "DATABASE SYSTEMS" である
G型関係<PROJ-SUBJ>を削除する場合、DELETE文は次の様である。

```

RANGE (P,PROJECT);
RANGE (PJ,PROJ-SUBJ);
RANGE (J,SUBJECT);
DELETE PJ
WHERE P.PNO = "C858" AND
      J.SUBJECTN = "DATABASE SYSTEMS";
GO;

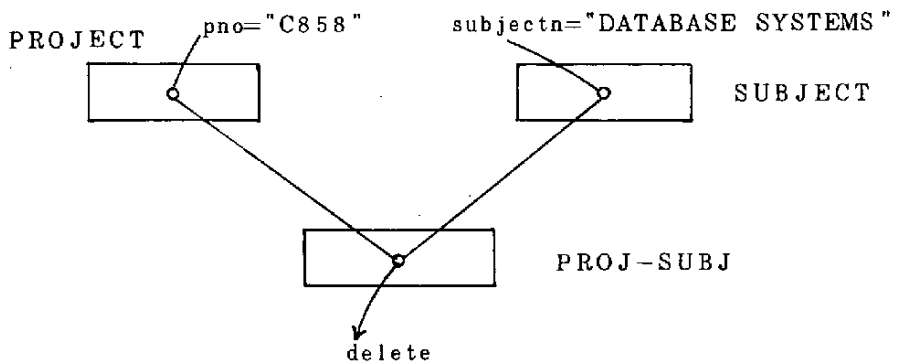
```

LCS

PROJ-SUBJ (G型関係)

@ P	@ J
(MMUI80)	(DATABASE...)
(C858)	(DATABASE...) ← 削除

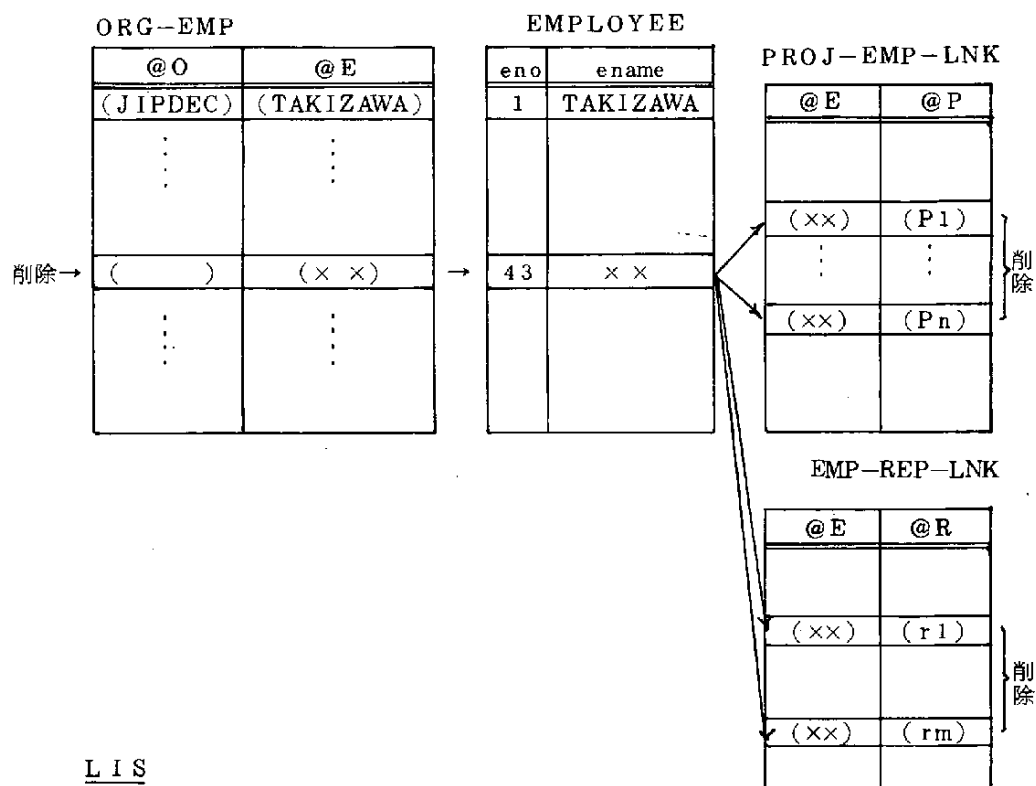
LIS



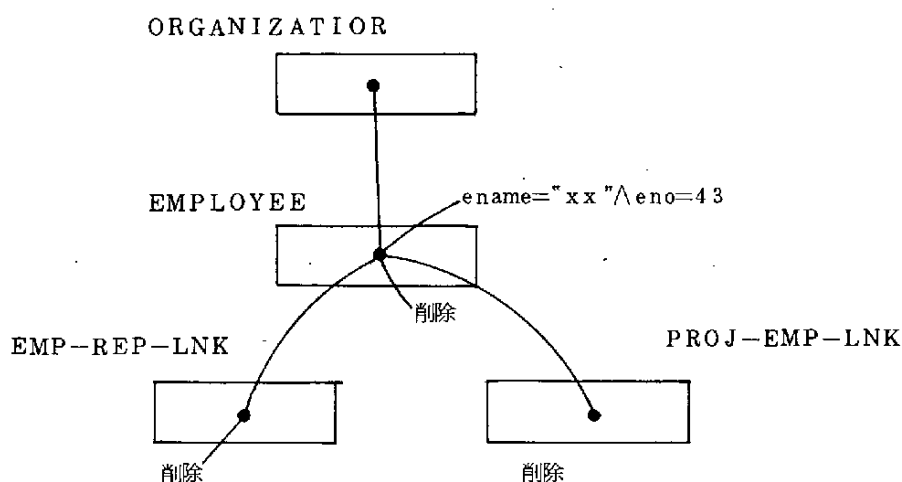
(IV) 従業員(名前="xx"、従業員番号=43)の組をORG-EMPから削除する場合、
DELETE文は次の様である。

```
RANGE (E,EMPLOYEE)(OE,ORG-EMP);
DELETE OE WHERE E.ENO=43 AND E.OE=OE AND E.ENAME="XX";
```

LCS



LIS



F. GO コマンド

〔フォーマット〕 <go-c> ::= go (<qgf #>);

〔機能〕 問合せ文 (RETRIEVE, APPEND, REPLACE, DELETE, OUTPUT) を実行する。

〔解説〕 利用者の入力した問合せ文を実行する。行番号 <qbf #> の指定の有無によって、実行される問合せ文が異なる。

(1) go ;

直前に入力した問合せ文 (RETRIEVE, APPEND, DELETE, REPLACE) を実行する。

(2) go n ;

QBFR内の指定行 n から始まる問合せ文 (RETRIEVE, APPEND, DELETE, REPLACE)、又は OUTPUT コマンドを実行する。

〔例〕

```
#00030 RANGE (P,PROJECT);  
#00040 RETRIEVE INTO R1 (P.PND);  
#00050 RANGE (O,ORGANIZATION);  
#00060 RETRIEVE INTO R2 (O.ORG-NAME);  
#00070 GO;
```

← 直前の問合せ
R 2 が実行さ
れる。

RETRIEVE INTO R2 (ORG-NAME=O.ORG-NAME) ;

```
THE ELAPSE TIME FOR RETRVP,QM.. 66  
THE ELAPSE TIME FOR GTG..... 13  
THE ELAPSE TIME FOR RQGP..... 11  
THE ELAPSE TIME FOR DQGP..... 17  
THE ELAPSE TIME FOR DFA1..... 2  
THE ELAPSE TIME FOR DML..... 1  
THE ELAPSE TIME FOR DMLG..... 1245
```

R 1 を実行するには、R 1 の行番号を指定した GO コマンドを実行する。

#00100 DISP;

LINE1 STATEMENT

40IRETRIEVE INTO R1 (P.PNO);
60IRETRIEVE INTO R2 (O:ORG-NAME);

#00110 GD 40;

RETRIEVE INTO R1 (PNO=P.PNO) ;

THE ELAPSE TIME FOR RETRVP,OM.. 47
THE ELAPSE TIME FOR GTG..... 10

G. STATUS コマンド

[フォーマット] <status-c> :: = status { <rrel-name> } ;

[機能] 問合せの実行状況を画面に表示する。

[解説] 直前に実行した問合せの場合、結果関係名<rrel-name>を省略することができる。

問合せの実行状況としては、次のメッセージが表示される。

(1) 実行メッセージ

#00270 STAT;
Y1 IS EXECUTING

(2) 正常終了(結果無し)メッセージ

#00620 STATUS;
Y2 IS COMPLETED, NO RESULT

(3) 正常終了(結果有り)メッセージ

#00280 STAT;
Y1 IS COMPLETED, WAITING FOR OUTPUT
RETRIEVE OCC. =000000010

↑ 検索結果数

(4) 異常終了メッセージ

#00990 STAT;
Y3 ABNORMALLY ENDED

H. OUTPUTコマンド

```
[フォーマット] <output-c> ::= output <rrel-name>
                                     [ ( <raformat> [ <raformat> ) ] ];
                                     <raformat> ::= <ratt> [ / <format> ]
                                     <format> ::= <type> / <length>
                                     <type> ::= C | I
```

〔機能〕 RWS内の結果関係<rrel-name>を指定されたフォーマット<raformat>で出力する。

〔解説〕 問合せの実行が正常終了した結果関係<rrel-name>をRWSから読み出し（実行が正常終了したかどうかは、STATUSコマンドで確認する）、指定されたフォーマットで端末画面に表示する。この時、冗長な組は除かれる。

出力フォーマット内で指定される結果属性<ratt>は結果関係内に存在するものでなければならない。出力できるのは、利用者が実行した結果関係だけである。他の利用者の結果関係は出力できない。

〔例〕

```
#00240  HELP RREL;
```

```
(REL-NAME=R, DEGREE= 3, CARDINALITY= 6, WIDTH= 28,
TPMODE=0, FLAG=0, EXIST=1, ENDCODE=0000 )
```

1	AT#	1	ATT-NAME	1	ITYP	1	LN	1	NGTH	1	CARDINLTY	1	ITYP	1	DOMAIN
1	1	1	PND	1	C	1	8	1	1	1	0	1	PND	1	
1	2	1	ENAME	1	C	1	10	1	6	1	0	1	ENAME	1	
1	3	1	BIRTH-PLACE	1	C	1	10	1	3	1	0	1	BIRTH-PLACE	1	

```
#00250  OUTP R (PND/C8, ENAME/C10, BIRTH-PLACE/C10);
```

```
** R **
* RESULT CARDINALITY = 000000006
```

1	PND	1	ENAME	1	BIRTH-PLAC
1	MMUI80	1	IACKI	1	ITOCHIGI
1	MMUI80	1	IKOSEKI	1	IMIYAGI
1	MMUI80	1	ISUZUKI	1	ITOKYO
1	MMUI80	1	ITAKIZAWA	1	ITOKYO
1	MMUI80	1	ITSUKAMOTO	1	ITOKYO
1	MMUI80	1	IYOKOTSUKA	1	ITOKYO

I. MODE CHANGE コマンド

$$\begin{aligned} \{ \text{フォーマット} \} \quad \langle \text{mode-c} \rangle &::= \underline{\text{mode}} \langle \text{rel-name} \rangle \langle \text{mode} \rangle ; \\ \langle \text{mode} \rangle &::= T \mid P \\ &\begin{cases} T = \text{Temporary} \\ P = \text{Permanent} \end{cases} \end{aligned}$$

〔機能〕 結果関係の関係作業領域（RWS）への格納モードの変更を行う。

〔解説〕 格納モードにより結果関係はLDPセッション後もRWS内に保存、又は消去される。

j) mode = P

LDPセッション終了後（LOGOFF後）もRWS内に保存される。

ii) mode = T

LDPセッション終了時に、RWSから消去される。

結果関係は、Temporaryモードで生成されるので、結果関係を保存する為には、modeをPに変更する。

変更できるのは利用者が実行した結果関係の格納モードだけである。但し、異常終了した結果関係をPermanentモードに変更することはできない。

〔例〕

i) 結果関係名 R1 のモードを P に変更する場合、MODEC コマンドは次の様である。

#00120 MODEC R1 P;
THE MODE OF R1 WAS CHANGED BY P

ii) 結果関係名 R1 のモードを T に変更する場合、MODEC コマンドは次の様である。

#00130 MODEC R1 T;
THE MODE OF R1 WAS CHANGED BY T

J. ERASE コマンド

```
[フォーマット] <erase-c> ::= erase<rrel-name>;
```

〔機能〕 結果関係<rrel-name>を関係作業領域 (RWS) から消去する。

〔解説〕 関係作業領域に存在する結果関係<rrel-name>を消去する。但し、以下の場合は消去されない。

i) 結果関係を生成する問合せが実行中の場合。

ii) 結果関係の格納モードがPermanentの場合。

Permanent モードの結果関係を消去する場合は、MODE CHANGE コマンドで Tem

poraryモードに変更後に実行する。

削除できる結果関係は、利用者が実行した問合せに対する結果関係だけである。

〔例〕 結果関係 R 2 の削除は、次の様である。

```
#00190  ERASE R2;  
R2      WAS ERASED
```

K. EDIT コマンド

〔フォーマット〕 <edit-c> ::= edit <qb#>

〔機能〕 Q B F R 内の指定行<qb#>の内容を更新（追加、削除、変更）する。

〔解説〕 更新は、edit コマンド入力後のプロンプト "><qb#>" の後に入力された内容で実行される。

(1) 追加

追加は、Q B F R 行間の使用されていない行番<qb#>を指定する。編集行で入力された内容が、指定位置<qb#>に追加される。

“;” は入力の終了を表わすので、更新内容として入力するには 2 個以上連ねて入力する。

(2) 削除

削除は、削除する Q B F R 行の番号<qb#>を指定する。編集行には、空行を入力する。

(3) 変更

変更は、変更する Q B F R 行の番号<qb#>を指定する。編集行には、変更する内容を入力する。

〔例〕

```
#00120  DISPLAY;
```

```
LINE|  STATEMENT
```

```
-----|-----  
60|RETRIEVE INTO RR (E.END,P.PND)
```

```
70|WHERE
```

```
80|  E.QE = PE.QE AND PE.QP = P.QP;  
-----|-----
```

```
#00130  EDIT 60;
```

```
>00060  RETRIEVE INTO RR (E.END, E.ENAME,P.PND);
```

↑
行番号 6 0 の変更イメージの入力

```
#00140 EDIT 75;
)00075 E.END LE 10 AND;
```

行番号 7 0 と 8 0 の間に条件を追加する。上記の変更結果は次の様である。

```
#00150 DISPLAY;
```

```
LINE1 STATEMENT
```

```
-----|
60IRETRIEVE INTO RR (E.END, E.ENAME,P.PNO) |
```

```
70IWHERE |
```

```
75I E.END LE 10 AND |
```

```
80I E.OE = PE.OE AND PE.OP = P.OP; |
-----|
```

行番号 7 5 を削除する場合の E D I T コマンドは、次の様である。

```
#00160 EDIT 75;
```

```
)00075 ;
```

削除結果は、次の様である。

```
#00170 DISPLAY;
```

```
LINE1 STATEMENT
```

```
-----|
60IRETRIEVE INTO RR (E.END, E.ENAME,P.PNO) |
```

```
70IWHERE |
```

```
80I E.OE = PE.OE AND PE.OP = P.OP; |
-----|
```

L. DISPLAY コマンド

〔フォーマット〕 <display-c> ::= display [<qbf#>{ , <qbf#> }] ;

〔機能〕 Q B F R の内容を表示する。

〔解説〕

(1) display ;

Q B F R の内容を全て表示する。

(2) display <qbf#> ;

指定された行番号 <qbf#> に該当する Q B F R の内容を表示する。

(3) display <qbf#1> , <qbf#2> ;

指定範囲 (<qbf#1> から <qbf#2> まで) の Q B F R の内容を表示する。

〔例〕

i)

#00200 DISP;

LINE1 STATEMENT

```
-----|
40IRETRIEVE INTO R1 (P.PNO);
60IRETRIEVE INTO R2 (O.ORG-NAME);
140IOUTP R1 (PNO/C8);
150IOUTP R2 (ORG-NAME/C30);
170IOUTP R2 (ORG-NAME/C30);
-----|
```

ii)

#00230 DISP 40;

LINE1 STATEMENT

```
-----|
40IRETRIEVE INTO R1 (P.PNO);
-----|
```

iii)

#00220 DISP 40,60;

LINE1 STATEMENT

```
-----|
40IRETRIEVE INTO R1 (P.PNO);
60IRETRIEVE INTO R2 (O.ORG-NAME);
-----|
```

M. RENUMBER コマンド

〔フォーマット〕 <renumber-c> ::= renumber;

〔機能〕 Q BFR の行番号を 10 から 10 単位に renumber する。

〔解説〕 問合せ作成の作業領域 (Q BFR) 内の行番号 (Q BFR #) を 10 から、10 単位の間隔で番号をつけなおす。

Q BFR 内には、RETRIEVE, APPEND, REPLACE, OUTPUT, define, OUTPUT コマンドだけが残され、他のコマンドは消去されている。

N. DFS コマンド

〔フォーマット〕 <dfa-c> ::= dfa<i>;

<i> ::= 0 | 1 | 2 | 3

〔機能〕 最適化のサーチレベルを指定し、アクセスパス生成で指定されたレベルまでサーチする。

〔解説〕

(1) DFA0 ;

この場合、利用者がLDPから与えられた情報を元にアクセスパスを決定する。

(2) DFA1 ;

最適化のサーチレベルを1レベルとする。

(3) DFA2 ;

最適化のサーチレベルを2レベルとする。

(4) DFA3 ;

最適化のサーチレベルを3レベルとする。

O. HELP コマンド

```
〔フォーマット〕 <help-c> ::= help lcs ; |  
                        help rel { <lcs-name> } ; |  
                        help rrel { <rrel-name> } ; |  
                        help var { <var> } ;
```

〔機能〕 LDP内に保持している情報を利用者に表示する。

〔解説〕 フォーマットにより、次の情報が利用者に表示される。

i) help lcs ;

LDPのHI情報内に登録されているLCS名を表示する。

ii) help rel ;

QTPコマンドで指定されたLCS<lcs-name>内に定義されている関係の関係名、属性構成(属性名、タイプ、桁数等)の情報を表示する。関係名<rel-name>を指定した場合、LCS内の指定された関係の属性構成だけが表示される。

iii) help rrel ;

RWS内に格納されている利用者の結果関係の属性構成が表示される。QTPコマンドで指定されたLCS以外のLCSに対する結果関係は表示されない。結果関係<rrel-name>が指定された場合、指定された結果関係の属性構成だけが表示する。

iv) help var ;

現在までに定義した組変数の情報を表示する。組変数名<var>が指定された場合、指定された組変数の情報だけを表示する。

〔例〕

i) HELP LCS;

```
#00040  HELP LCS;
LCSDCDBS      AT M170-F
LCSLRSSDB     AT M170-F
LCSFROJDB     AT M170-F
```

ii) HELP REL PROJECT;

#00060 HELP REL PROJECT;

ITYPE	REL-NAME	IAT#	IATT-NAME	IES#	ITILENGTH	ITG#	ITARGE-NAME	IRIM
IESR	PROJECT	1	1PNO	01C	8	0	PROJECT	1101
IESR	PROJECT	2	1PNO	01C	8	1	PROJECT	1010
IESR	PROJECT	3	1PNAME	01C	100	2	PROJECT	1010
IESR	PROJECT	4	1PROJ-TYPE	01C	1	3	PROJECT	1010
IESR	PROJECT	5	1STATUS-F	01C	3	4	PROJECT	1010
IESR	PROJECT	6	1YEAR	01I	2	5	PROJECT	1010
IESR	PROJECT	7	1YEAR	01I	2	6	PROJECT	1010
IESR	PROJECT	8	1BUDGET	01I	11	7	PROJECT	1010

iii) HELP RREL;

#00240 HELP RREL;

(REL-NAME=R, DEGREE= 3, CARDINALITY= 6, WIDTH= 28,
TPMODE=0, FLAG=0, EXIST=1, ENDCODE=0000)

IAT#	IATT-NAME	ITYPE	ILNGTH	CARDINLTY	IATYP	IDOMAIN
1	1PNO	C	8	1	01PNO	
2	1ENAME	C	10	6	01ENAME	
3	1BIRTH-PLACE	C	10	3	01BIRTH-PLACE	

iv) HELP VAR;

```
#00120  HELP VAR;
NO=      1  VAR=E      REL-NAME=EMPLOYEE      TYPE=L  OPTV=  0  RTYPE=0
MARK=
NO=      2  VAR=PE     REL-NAME=PROJ-EMP-LNK    TYPE=L  OPTV=  0  RTYPE=1
MARK=
NO=      3  VAR=P      REL-NAME=PROJECT        TYPE=L  OPTV=  0  RTYPE=0
MARK=
```

4.4 VQP コマンド

〔フォーマット〕 <VQP-C> ::= VQP <view-schema-name>;

〔機能〕 HI(VI)から、ログオンされたLCSの下での<view-schema-name>の情報をLDP内へloadする。ログオンする視野として許された利用者のみが使用できる。

```
VQP <view-schema-name>;
{<QTP-command>;}
end;
```

A. VIP コマンド

```
<VIP-command> ::= <range-c> | <define-c> |  
                  <drop-c> | <destroy-c> |  
                  <store-c> | <edit-c> |  
                  <end-c> | <display-c>
```

VIP コマンドは、視野の定義や消去を行う為のコマンドである。

B. DEFINE コマンド

```
[フォーマット] <define-c> ::= define view <view-name>  
                               <view-def> <target-list>  
                               where <qual>  
                               [ : <view-def> <target-list>  
                               where <qual> ] ;
```

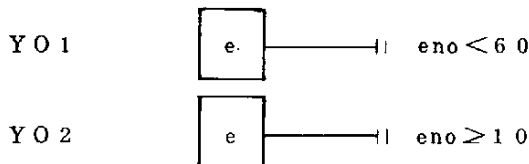
〔機能〕 LCS 関係 and/or 視野関係上に、新たに視野関係 <view-name> を定義し、視野管理情報 (VI) 内に定義情報を格納する。

〔解説〕 複数の視野定義の集合を、ビュー名 <view-name> で視野管理情報 (VI) 内に格納する。この時、視野内で参照されている組変数も VI 内に保持される。

視野内の全ての視野定義は、共通の組変数の集合を持つ。この為、視野内の視野定義間の主結合が可能である。

〔例〕 視野定義の例として、従業員番号が 60 未満の従業員番号 (eno)、従業員名 (ename)、出身地 (birth-place) を持つ視野定義 (Y01) と、従業員番号が 10 以上の従業員番号 (eno)、従業員名 (ename)、出身地 (birth-place) を持つ視野定義 (Y02) に対する視野スキーマ YOKO は、次の様に定義する。

```
RANGE (E,EMPLOYEE);  
RANGE (P,PROJECT);  
DEFINE VIEW YOKO  
  Y01 (E.ENO,E.ENAME,E.BIRTH-PLACE)  
    WHERE E.ENO LT 60;  
  Y02 (E.ENO,E.ENAME,E.BIRTH-PLACE)  
    WHERE E.ENO GE 10;
```



C. DROP コマンド

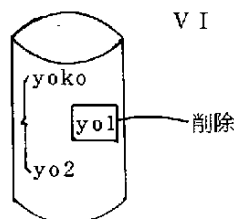
$$[\text{フオマツト}] \quad \langle \text{drop-c} \rangle ::= \underline{\text{drop}} \langle \text{view-def} \rangle [, \langle \text{view-def} \rangle \\ \text{in } \langle \text{view-name} \rangle] ;$$

〔機能〕 視野管理情報 (V I) 内の視野名<view-name>から視野定義<view-def>を削除する。

〔解説〕 視野内の不要になった視野定義<view-def>を視野管理情報(VI)から削除する。

例えば、視野YOKOから、視野定義YO1を削除する場合、dropコマンドは次の様である。

```
drop YOI in YOKO;
```



D. END コマンド

[7. -マ 7.] <end-c> ::= end ;

〔機能〕 LDPセッションを終了させる。この時、関係格納領域(RWS)に格納されている結果で、モードがTemporaryのものを全て削除する。

E. LOGOFFコマンド

〔フォーマット〕 <logoff-c> ::= logoff ;

〔機能〕 LDPの実行を終了させる。

〔解説〕 LDPの実行を終了させ、終了メッセージと終了時間を表示する。

4.5 VDEFINE コマンド

〔フナマツト〕 <Vdefine-c> ::= vdefine <view-schema-name>;

〔機能〕 新たに視野として視野スキーマ名を<view-schema-name>を定義する。

〔解説〕 視野スキーマ名は、HI内でユニークな16桁以内の名前を定義する。宣言した視野スキーマ名と同一名がHI(VI)内に登録されている場合、メッセージが表示される。

```
Vdefine <view-schema-name>;  
    {<DI P-command>;}  
  
end;
```

4.6 VUPDATE コマンド

〔フォーマット〕 <Vupdate-c> ::= Vupdate <view-schema-name>;

〔機能〕 HI(VI)から、ログオンされたLCSの下での視野スキーマの定義式をLDP内へloadする。視野に対して許可された利用者のみが利用できる。

```
vupdate <view-schema-name>;  
    {<VI P-command>;}  
  
end;
```

4.7 DESTROY コマンド

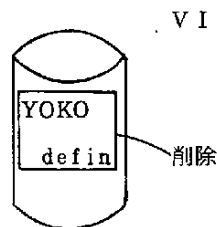
〔フォーマット〕 <destroy-c> ::= destroy <view-name> [, <view-name>];

〔機能〕 視野管理情報(VI)内の視野名<view-name>を削除する。

〔解説〕 不用になった視野を視野管理情報(VI)内から削除する。

例えば、視野YOKOを削除する場合、destroy コマンドは次の様である。

```
destroy YOKO;
```



5. LDPエラーメッセージ

各コマンドに対するエラーメッセージを以下に示す。

A. 一般

(1) COMMAND ERROR

これは、入力コマンドが正しいLDPコマンドではないことを示す。

(2) THIS COMMAND IS NOT BUILT IN

これは、入力コマンドとして実現していないコマンドであることを示す。

(3) INVALID CHARACTER... (x), ST# = n, IN <command-name>

これは、コマンド<command-name>文内に不正な文字xが現われたことを示す。

(4) INVALID CHARACTER(S) APPEAR IN <command-name>

これは、コマンド<command-name>文内に不正な文字が現われたことを示す。

(5) INVALID CHARACTER IN <command-name>

これは、コマンド<command-name>文内に不正な文字が現われたことを示す。

B. login

(1) YOUR USER-ID IS NOT AUTHORIZED

これは、LOGON文で指定した利用者が登録されていないことを示す。

(2) YOUR PASSWORD IS NOT AUTHORIZED

これは、LOGON文で指定した利用者のパスワードが誤っていることを示す。

(3) YOUR USER-ID IS ALREADY USED

これは、LOGON文で指定した利用者識別名がすでに使用されていることを示す。

(4) YOUR LCS-NAME IS NOT AUTHORIZED

これは、QTP文で指定したLCS名が登録されていない、あるいは利用権が利用者に与えられていないことを示す。

C. Retrieve

(1) SYNTAX ERROR IN QUAL

これは、条件式<qual>内にシンタックスの誤りがあったことを示す。

(2) <var>・<att-name> WAS NOT FOUND ATT RELATION

これは、属性<var>・<att-name>が属性テーブル(ATT)内に見つからなかったことを示す。

(3) OPERATOR MISSING

これは、条件式内で演算子が抜けていることを示す。

(4) DML BLOCK OVER

これは、DML情報がメモリ内のエリアをオーバーフローしたことを示す。

(5) STACK OVERFLOW ABEND

これは、スタックがオーバーフローしたことを示す。

(6) STACK UNDERFLOW ABEND

これは、スタックがアンダフローしたことを示す。

D. RANGE

(1) <rel-name> DOES NOT EXIST

これは、RANGE文で指定した関係<rel-name>がLCS内に存在しないことを示す。

(2) <var> WAS NOT FOUND IN RTBL

これは、組変数<var>が組変数テーブル(RTBL)内に見つからなかったことを示す。

(3) <var> IS ALREADY DECLARED

これは、組変数<var>が既に宣言されていることを示す。

E. append

(1) <att-name> IS PRIMARY KEY

これは、APPEND文又はREPLACE文のtarget-list内で主キーに値を設定したことを示す。

(2) <att-name> IS NOT SPECIFIED

これは、APPEND文のtarget-list内にNNA属性の属性に対する値の設定がないことを示す。

F. replace

(1) <att-name> IS A CALC ITEM

これは、REPLACE文でCALC項目の値を変更しようとしたことを示す。

G. EDIT

(1) LINE-NO=nn WAS NOT FOUND IN QBFR

これは、指定された行番号が作業領域(QBFR)内に見つからないことを示す。

H. Status

(1) <rrel-name> IS COMPLETED, NO RESULT

これは、STATUS文に対して、問合せの実行は終了したが、結果関係<rrel-name>に解がなかったことを示す。

(2) <rrel-name> IS COMPLETED, RESULT OVERFLOW

これは、STATUS文に対して、問合せの実行は終了したが、結果関係<rrel-name>の組をRWSに格納中にオーバーフローが発生したことを示す。

(3) <rrel-name> IS ABNORMALLY ENDED

これは、結果関係<rrel-name>の実行が異常終了したことを示す。

I. OUTPUT

(1) <rrel-name> IS NOT FOUND IN SYSTEM STORAGE

これは、指定した結果関係<rrel-name>がRWS内に存在しないことを示す。

(2) <rrel-name> IS NOT STORED YET

これは、OUTPUT文に対して検索結果がRWSにまだ格納されていないことを示す。

(3) <att-name> IS NOT FOUND

これは、OUTPUT文で指定された結果属性<att-name>が結果関係内に存在しないことを示す。

J. erase

(1) <rrel-name> IS PERMANENT MODE

これは、ERASE文で指定した結果関係のモードがPermanentモードであることを示す。

K. grant

(1) YOU ARE NOT AUTHORIZED

これは、GRANT文を実行した利用者がLCSの管理者でないことを示す。

(2) USER-ID IS ALREADY EXIST……(<user-id>)

これは、GRANT文で指定した利用者にすでに利用権が与えられていることを示す。

(3) LCS-NAME=(<lcs-name>) IS NOT FOUND IN HI

これは、GRANT文で指定したLCS名がHIに登録されていないことを示す。

L. go

(1) <rrel-name> WAS ALREADY IN SYSTEM STORAGE

これは、GOコマンドで実行しようとした問合せと同じ結果関係名がすでにRWSに存在することを示す。

6. HIへの問合せの実行

HI及びRWS情報は、CODASYL DBS（具体的にはAIMで実現）としてデータベース内に格納されている。このHIが格納されているデータベースに対して、LDPで問合せを作成し、アクセスすることができる。HIへの問合せの実行は、QTPコマンドでLCS名として、HIのデータベースに対するLCS名（LCSLRSSDB）を入力する。QTPコマンドは、他のデータベースをアクセスするのと同じである。

〔例〕 HIに格納されているLCS名の検索は次の様に実行される。

```

*** JDBES LDP-V3 START... 15.00.10.377

#00010 LOGON LDP YOKOM/MINOR;
LDP IS LOGONED
#00020 QTP LCSLRSSDB;
LCSLRSSDB IS OPENED
#00030 RANGE (H,HI);
#00040 RETRIEVE INTO TT (H,LCSNAME);
#00050 GO;

RETRIEVE INTO TT ( LCSNAME=H.LCSNAME ) ;
THE ELAPSE TIME FOR RETRVP,QM.. 51
THE ELAPSE TIME FOR GTG..... 12

```

QTPコマンドでLCS名としてLCSLRSSDBを指定する。

LCS名の検索問合せ

実行結果は次の通りである。

```

#00110 OUTP TT (LCSNAME/C16);

** TT **
* RESULT CARDINALITY = 000000003

-----
|LCSNAME|
|-----|
|LCSDCDBS|
|LCSLRSSDB|
|LCSPROJOB|
|-----|

```

TTの出力

16 bytes

付記 2 GDP コマンド

全体データベースプロセッサ (GDP) の利用者インタフェース (GDP コマンド) を示す。

1. GDP 状態

図 1.1 と表 1.1 に GDP の状態を示す。

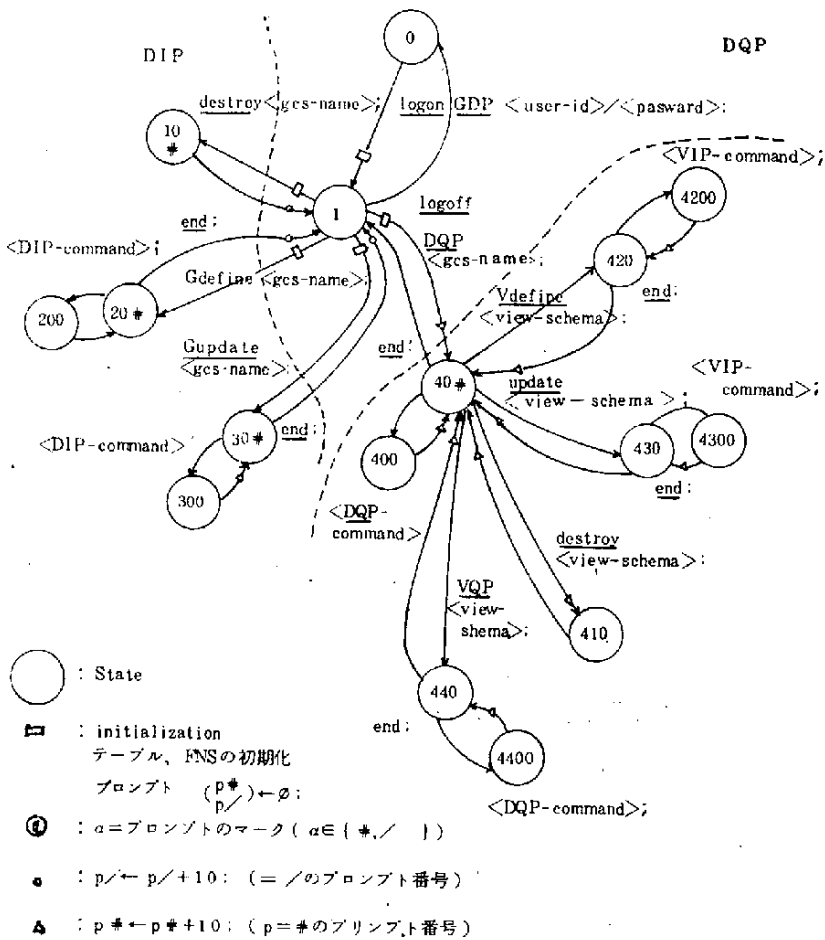


図 1.1 GDP の状態遷移

表 1.1 GDP の状態

state #	state の意味
0	initial state 全てのテーブルはclear されている
1	logged-on/ logonされGDPコマンドを待っている wait for GDP command (全てのテーブルはclear されている)
10	destroy 処理中
20	Gdefine 処理中/ wait for DIP command
200	DIP command 処理中
30	Gupdate 処理中/ wait for DIP command
300	DIP command 処理中
40	DQP command 処理中 wait for DQP/View command
400	DQP command 処理中 range. retrieve 等の処理中
410	Destroy command 処理中
420	Vdefine 処理中/wait for DIP command
4200	VIP command 処理中
430	Update 処理中/wait for DIP command
4300	VIP command 処理中
440	VQP command 処理中/ (viewに対するoperation の処理) wait for DQP command
4400	DQP command for view 処理中

1.1 GDP コマンドクラス

GDPへのコマンドは、次の4つのクラスからなる。

- (1) 第1クラス state 0 から 1 への遷移を行う。

logon GDP <user-id>/<password>; GDP sessionの
開設を行う。

- (2) 第2クラス state 1 から 2 けたの state への遷移を行う。

- i) DQP <gcs-name>; DQPへのアクセス
- ii) define <gcs-name>; <gcs>の定義
- iii) update <gcs-name>; <gcs>の更新
- iv) destroy <gcs-name>; <gcs>の消去

V) logoff ;

(RWSのclear)

GDP sessionを閉じる。

(3) 第3クラス 2ケタのstateから3ケタのstateへの遷移を行う。

- i-1) <DQP-command>; GCS問合せの入力
- i-2) Vdefine <view-schema>; 視野の定義を行う
- i-3) update <view-schema>; 視野の実行を行う
- i-4) destroy <view-schema>; 視野の消去
- i-5) VQP <view-schema>; 視野へのアクセスを行う
- ii-1)) <VIP-command> GCSの管理を行う
- iii-1)

(4) 第4クラス 3ケタのstate内の遷移を行う。

- i-2)) -1 <VIP-command> 視野の管理を行う
- i-3)

2. DIPインタフェース

DIPインタフェースは、GAにGCSの管理の為に、次の様にGCS定義、GCS更新、GCS
消去コマンドを提供する。

1) GCS定義

GCSは次の様に定義される。

```
logon GDP <user-id>/<password>;  
/9999 define <gcs-name>;  
新たに<gcs-name>のGCSを定義することを宣言する。  
#9999 <DIP-command>;  
      : } 繰り返し  
#9999 end;  
/9999
```

2) GCSの消去

GCSの消去は次の様に行う。

```
logon GDP <user-id>/<password>;  
/9999 destroy <gcs-name>;  
GCS<gcs-name>を消去する。  
/9999
```

3) GCSの更新

GCSのスキーム、定義式の変更は次の様にして行う。

```
logon GDP <user-id>/<password>;  
/9999 update <gcs-name>;  
更新するGCS<gcs-name>の定義式をDIPにロードする。  
#9999 <DIP-command>;  
      : } 繰り返し  
#9999 end;  
/9999
```

2.1 GCS定義コマンド

[フォーマット] <define-c> ::= define <gcs-name>;

[機能] 新たにGCSとしてGCS名<gcs-name>を宣言する。

〔解 説〕 GCSの定義に先立って、GCS名<gcs-name>を宣言する。DI内でGCS名はユニークでなければならない。GCS内のGCS関係は、<DIP-command>によって定義される。定義されたGCSの管理者 (administrator) は、ログオン時の利用者となる。

```
define <gcs-name>;
  { <DIP-command>; }
end;
```

2.2 GCS更新コマンド

〔フォーマット〕 <update-c> ::= update <gcs-name>;

〔機能〕 既に定義されているGCS<gcs-name>のGCS関係スキーム、定義式をDIP (QBFR内)にロードする。

〔解 説〕 GCSの更新は、<gcs-name>の管理者だけが許される。QBFR<DIP-command>の編集edit コマンドを用いて、GCS定義式の更新を行い、store コマンドで、DIへGCS定義を格納する。

```
update <gcs-name>;
  { <DIP-command>; }
end;
```

2.3 GCS消去コマンド

〔フォーマット〕 <destroy-c> ::= destroy <gcs-name>;

〔機能〕 既に定義されているGCS<gcs-name>をDIから消去する。

〔解 説〕 GCS<gcs-name>の消去は、管理者だけが許される。

2.4 DIPコマンド

```
<DIP-command> ::= <range-c> | <define-c> |
                  <edit-c> | <drop-c> |
                  <list-c> | <rlist-c> |
                  <grant-c> | <revoke-c> |
                  <change-c> | <define-u-c> |
                  <ulist-c> | <display-c> |
                  <renum-c> | <store-c> |
```

<del-c>

GCSの定義や利用者の定義などのDI情報を定義する為のコマンドである。

A. RANGEコマンド

〔フォーマット〕 <range-c> ::= range <range-def> { <range-def> } ;
 <range-def> ::= (<var-list> , <lrel-name> : <dm-id>)
 <var-list> ::= <var> | (<var> { , <var> })

〔機能〕 GCSの定義で参照するLCS内の関係<lrel-name>とそのサイト<dm-id>に対する組変数を宣言する。

〔解説〕 参照するLCSがあるサイト<dm-id>の指定が必要な以外は、QTPコマンドのRANGEコマンドと同じである。

B. DEFINEコマンド

〔フォーマット〕 <define-c> ::= define <grel-name>
 <gatt-list> <g-def>;
 <gatt-list> ::= (<ga-def> { , <ga-def> })
 <ga-def> ::= <gatt-name> [/ [<domain>] [/ <format>]]
 <format> ::= <type> <length>
 <type> ::= I | C | P
 {
 I : integer
 C : character
 P : primary key
 <g-def> ::= <sub-def> | <sub-def> <sop> <sub-def> |
 (<sub-def>)
 <sub-def> ::= (<target-list>) [where <qual>]
 <sop> ::= : * | : + | : -
 ($\cap$) ($\cup$)

〔機能〕 GCS内のGCS関係をLCS関係を用いて定義する。

〔解説〕 参照するLCS関係は、新じめRANGEコマンドで宣言されなければならない。
GCS名はDI内でユニークである。同一名のGCSがDI内に既に存在する場合、エラーメッセージが表示される。

属性定義<ga-def>で定義域<domain>、フォーマット<format>が省略された場合、LCS関係内の対応する属性の定義域・フォーマットをとるものとする。

〔例〕 従業員番号が5未満で、レポートを書いている従業員の名前とレポート番号を属性として持つGCS関係の定義は、次の様である。

GCS名の宣言 組変数の宣言

```

/00010  DEFINE GCSNAME7;
#00010  RANGE(E,EMPLOYEE:3)(ER,EMP-REP-LNK:3)(R,REPORTR:3);
#00020  DEFINE GRELNAME7
#00030      (REPNO/NUMB/C8, ENAME/ENAME/C20)
#00040      (REPNO=R.RNO, ENAME=E.ENAME) WHERE
#00050      E.END < 5 AND E.QE=ER.QE AND ER.QR=R.QR;

```

GCS関係の定義

C. DROP コマンド

〔フォーマット〕 <drop-c>::=drop <grel-name>[, <grel-name>];

〔機能〕 GCS定義からGCS関係<grel-name>を消去する。

D. RLIST コマンド

〔フォーマット〕 <rlist-c>::=rlist <r-par>;
 <r-par>::=RANGE | RREL | RATT

〔機能〕 <r-par>で指定される情報を画面に表示する。

〔解説〕 <r-par>の指定により、次の情報が画面に表示される。

i) rlist RANGE;

LCS関係に対して宣言された組変数の情報(LRTBLの内容)を画面に表示する。

ii) rlist RREL;

GCS内に定義されたGCS関係の情報(RRTBLの内容)を画面に表示する。

iii) rlist RATT;

GCS関係内に定義された属性の情報(RATBLの内容)を画面に表示する。

E. EDIT コマンド

〔フォーマット〕 <edit-c>::=edit <qbf #>;

〔機能〕 GCS定義の作業領域(QBFR)内の指定行<qbf #>単位の編集(追加、置換、消去)する。

F. LIST コマンド

〔フォーマット〕 <list-c>::=list [<qbf #> [, <qbf #>]];

〔機能〕 作業領域(QBFR)の内容を表示する。

〔解 説〕 作業領域の行番号<qbf#>の指定が無い場合は、作業領域の内容を全て画面に表示する。

行番号<qbf#>の指定がある場合、指定行、あるいは行指定範囲を画面に表示する。

G. DISPLAY コマンド

〔フォーマット〕 <display-c> ::= display <lcs-name>[{ , <lrel-name> }];

〔機 能〕 LCS関係のスキーマ情報 (e. g. 関係名、属性構成) を表示する。

〔解 説〕

i) display ;

LCS内の全てのLCS関係について、関係名、{ 属性名/定義域/フォーマット } を表示する。

ii) display L1, ..., Ln ;

LCS内のL1, ..., Ln 関係のスキーマ情報を表示する。

H. GRANT コマンド

〔フォーマット〕 <grant-c> ::= grant [<gcs-name>{ , <access-list> }]
to <userid>{ , <userid> } ;
<access-list> ::= ALL * | Retrieve | Append |
Delete | Replace

〔機 能〕 GCSの利用権を利用者に与える。

〔解 説〕 既に定義されているGCSの利用権を利用者に与え、利用者がGCSに対して問合せを実行出来る様にする。

<access-list>は、GCSに対して利用者のアクセスとして検索、更新(追加・変更・削除)のいずれかが許されるかを与えるものである。<access-list>を省略した場合、これらの全てのアクセス権を利用者に与える。

GRANT コマンドは、全体管理者(GA)だけが使用できる。

I. REVOKE コマンド

〔フォーマット〕 <revoke-c> ::= revoke <user-id>{ , <user-id> }
[from <gcs-name>] ;

〔機 能〕 利用者からGCSの利用権をはく奪する。

〔解 説〕 GCS名の指定がある場合は、指定されたGCSの利用権のみはく奪される。

又、GCS名が省略される場合は、利用者に与えられていた全てのGCSの利用権をなく奪する。

REVOKEは、全体管理者(GA)だけが使用できる。

J. CHANGEパスワードコマンド

[フォーマット] <change-c>::=change <user-id>/<password>
to <password>;

[機能] 利用者のパスワードを変更する。

[解説] CHANGEコマンドは、全体管理者(GA)だけが使用できる。

K. DEFINE USERコマンド

[フォーマット] <define-u-c>::=define <l-name>, <fname>
<userid>/<password>[to <dm-id>];

[機能] GCSへ利用者情報(利用者識別名、パスワード等)の登録を行う。

[解説] GCSを利用する利用者の定義を行う。GCSへのアクセス権はGRANTコマンドによって、利用者に与えられる。

利用者識別名は7桁以内、パスワードは5桁以内で定義される。又、<l-name>(名前)及び<fname>(名字)は各々20桁以内で定義される。

DEFINE USERコマンドは、全体管理者(GA)だけが使用できる。

L. ULISTコマンド

[フォーマット] <ulist-c>::=ulist [<gcs-name>];

[機能] GCSを利用できる利用者情報(利用者識別名、パスワード、利用者名、アクセスリスト)を画面に表示する。

[解説] 指定したGCSの利用権をもつ利用者のリストを表示する。全体管理者(GA)のみが使える。

GCS<gcs-name>の指定が無い場合、全てのGCSに対して利用者リストを表示する。

M. RENUMBERコマンド

[フォーマット] <renumber-c>::=renumber;

[機能] 作業領域(QBFR)内の行番号を10から、10単位で番号を付け直す。

N. STOREコマンド

[フォーマット] <store-c>::=store;

〔機 能〕 作業領域 (Q B F R) 内の G C S 定義式を D I (分散情報) に格納する。

〔解 説〕 作業領域 (Q B F R) 内の G C S 定義式内に 1 つでも誤りがあれば、D I への格納を行わない。誤りを修正して、再び store コマンドを実行する。

S T O R E コマンドで格納する G C S が既に D I 内に存在する場合、格納されている G C S 定義を消去してから、作業領域 (Q B F R) 内の G C S 定義情報を格納する。

3. DQPインタフェース

DQPインタフェースは、利用者にGCSへの問合せ及び視野の管理、視野への問合せを行うDQPコマンドを提供する。

DQPコマンドは、次の様に使用される。

```
logon GDP <user-id>/<password>;  
/9999 DQP <gcs-name>;
```

利用者が参照するGCS<gcs-name>を宣言する。これにより、DQPセッションが開始される。DQPセッション内では、次の様な処理を実行することができる。

(1) GCSへの問合せ

```
{ #9999 <DQP-command>; }
```

GCSへの検索問合せを定義し、実行させる。

(2) 視野への問合せ

```
#9999 VQP <view-schema>;
```

参照する視野<view-schema>の情報をDQPにロードする。

```
{ V9999 <VQP-command>; }
```

```
V9999 end;
```

(3) 視野の定義

```
#9999 Vdefine <view-schema>;
```

視野<view-schema>を新たに定義することを宣言する。

```
{ V9999 <VIP-command>; }
```

```
V9999 end;
```

(4) 視野スキーマの更新

```
#9999 Vupdate <view-schema>;
```

視野スキーマ<view-schema>を更新するため視野の定義式をDQPへロードする。

```
{ V9999 <VIP-command>; }
```

```
V9999 end;
```

(5) 視野スキーマの消去

```
#9999 Vdestroy <view-schema>;
```

視野スキーマ<view-schema>をDIから消去する。

(6) DQPセッションの終了

```
#9999 end;
```

DQPセッションを終了する。

3.1 DQP主コマンド

```
<DQP -mc> ::= <DQP -cc> | <VQP -c> |  
                <Vdefine -c> | <update -c> |  
                <destroy -c> | <end -c>
```

DQP主コマンドは、GCS又は視野に対するアクセスを行う為のコマンドである。

3.2 DQPコマンド

〔フォーマット〕 <DQP -cc> ::= DQP <gcs-name>;

〔機能〕 GCSに対するアクセスを行う為、必要なDI(GCSのスキーマ情報)をGDP内にloadする。

```
DQP <gcs-name>;  
    { <DQP -command> |  
      <VQP -command> |  
      <Vdefine -command> |  
      <update -command>; }  
  
end;
```

3.3 VQPコマンド

〔フォーマット〕 <VQP -c> ::= VQP <view-schema-name>;

〔機能〕 DI(VI)から、ログオンされたGCSの下での<view-schema-name>の情報をGDP内へloadする。ログオンする視野として許された利用者のみが使える。

```
VQP <view-schema-name>;  
    { <DQP -command>; }  
  
end;
```

3.4 VDEFINEコマンド

〔フォーマット〕 <vdefine -c> ::= vdefine <view-schema-name>;

〔機能〕 新たに視野スキーマを定義する為、視野スキーマ名の宣言を行う。

〔解説〕 宣言した視野スキーマ名<view-schema-name>と同一名の視野スキーマが既にDI(VI)内に登録されている場合、エラーメッセージが表示される。視野スキーマを定義した利用者が、この視野スキーマの管理者となる。視野スキーマの定義は、<VIP-

command>を用いて定義する。

```
Vdefine <view-schema-name>;  
  {<VIP-command>;}  
end;
```

3.5 VIEW UPDATE コマンド

〔フォーマット〕 <update-c>::=update <view-schema-name>;

〔機能〕 既にDI(VI)に登録されている視野スキーマ<view-schema-name>を更新する為、GDP内に定義情報をloadする。

〔解説〕 視野スキーマ定義を更新できるのは、視野スキーマを定義した利用者だけである。
視野スキーマの変更は、<VIP-command>を用いて実行する。

```
update <view-schema-name>;  
  {<VIP-command>;}  
end;
```

3.6 DESTROY コマンド

〔フォーマット〕 <destroy-c>::=destroy <view-schema-name>;

〔機能〕 DI(VI)から視野スキーマ<view-schema-name>の消去を行う。

〔解説〕 視野スキーマを削除できるのは、視野スキーマを定義した利用者だけである。

3.7 DQP コマンド

```
<DQP-c>::=<range-c> | <retrieve-c> |  
          <go-c>      | <edit-c>      |  
          <display-c> | <define-c>    |  
          <output-c>  | <modex-c>    |  
          <erase-c>   | <rename-c>   |  
          <status-c>  | <renumber-c> |  
          <rlist-c>   | <copy-c>
```

GCSに対する問合せを作成し、実行させる為のコマンドである。GCSの利用権をもつ利用者が使用する事ができる。

A. RANGE コマンド

[フォーマット] <range-c> ::= range <range-def>(<range-def>);
<range-def> ::= (<var>, <grel-name>)|
((<var>{,<var>}), <grel-name>)

[機能] GCS内の参照するGCS関係<grel-name>に対する組変数を宣言する。

[解説] RANGE コマンドは、QTP コマンドのRANGE コマンドと同じ様に定義される。参照するGCS関係<grel-name>は、DQP コマンドで宣言したGCS内に存在しなければならない。

[例] GCS関係GRELNAME7に対して、組変数R7を定義する場合、RANGE コマンドは次の様である。

```
#00010 RANGE (R7, GRELNAME7);
```

B. RETRIEVE コマンド

[フォーマット] <retrieve-c> ::= retrieve [into <rrel-name>]
(<target-list>)[where <qual>];

[機能] GCSに対する検索問合せを作成する。

[解説] 問合せに対するRETRIEVE文は、QTPインタフェースでのRETRIEVE文と同様に定義される。

[例] GCS関係GRELNAME7の中から、従業員名が" TAKIZAWA "である組のレポート番号と従業員名を求める場合、RETRIEVE コマンドは、次の様である。

```
#00020 RETRIEVE INTO TEMP (R7.REPNO, R7.ENAME)  
#00030 WHERE R7.ENAME = "TAKIZAWA";
```

C. GO コマンド

[フォーマット] <go-c> ::= go [<qbf#>];

[機能] retrieve 文、又はdefine 文を実行する。

[解説] 行番号<qbf#>の指定がある場合は、指定行から始まる retrieve 文、output、又はdefine 文を実行する。

行番号指定が無い場合、直前の作業領域(QBFR)内の retrieve 文、又はdefine 文を実行する。

G. MODE CHANGE コマンド

[フォーマット] `<modec-c> ::= modec <rrel-name> <mode>;`

`<mode> ::= T | P`
$$\left\{ \begin{array}{l} \text{T : Temporary} \\ \text{P : Permanent} \end{array} \right.$$

[機 能] 結果関係の関係作業領域 (RWS) への格納モードの変更を行う。

[解 説] 格納モードが T (temporary) の場合、セッション終了時に、RWS から消去される。格納モードが、P (ermanent) の場合は保存される。

格納モードを変更できるのは、利用者が作成した結果関係についてだけである。

H. DEFINE コマンド

[フォーマット] `<define-c> ::= define <view-rel> (<target-list>) [where <qual>];`

[機 能] DQP のセッション内で利用する視野関係の定義を行う。

[解 説] DQP のセッション内でのみ利用できる一時的な視野関係の定義を行う。もし同名を持つ視野関係が既に定義されている場合、利用者に通知し、古いものを消すかどうか選択させる。

I. RENUMBER コマンド

[フォーマット] `<renumber-c> ::= renumber;`

[機 能] 作業領域 (QBFR) 内の行番号 <qbf#> を 10 から、10 単位で番号を付け直す。

J. DISPLAY コマンド

[フォーマット] `<display-c> ::= display [ <grel-name> { , <grel-name> } ] ;`

[機 能] GCS 関係のスキーマ情報 (e. g 関係名、属性構成) を表示する。

[解 説]

i) display;

<view>, <gcs-name> の全ての <grel-name> について関係名、{ 属性名 / 定義域 / フォーマット } を表示する。

ii) display $G_1, \dots, G_n$;

<gcs>/<view>内の $G_1, \dots, G_n$ 関係のスキーマ情報を表示する。

K. RLIST コマンド

[フォーマット] <rlist-c> ::= rlist <r-par>;

<r-par> ::= RANGE | RREL | RATT

[機能] GCSに対して宣言した組変数の情報 (GRTBL)、問合せの情報 (GRR TBL)、問合せの属性構成 (GRATBL)を表示する。

[解説]

i) rlist RANGE ;

GCS内のGCS関係に対して宣言した組変数の情報 (GRTBL)を表示する。

ii) rlist RREL ;

GCSに対する問合せの情報 (GRR TBL)を表示する。

iii) rlist RATT ;

GCSに対する問合せの属性構成情報 (GRATBL)を表示する。

L. COPY コマンド

[フォーマット] <copy-c> ::= copy <qbf#>, <qbf#><pos><qbf#>;

<pos> ::= after | before

[機能] 作業領域 (QBFR)の内容を指定行の前、又は後にコピーする。

[解説] 指定された範囲の作業領域 (QBFR)の内容を指定行の前、又は後にコピーする。

i) copy q_1, q_2 after q_3 ;

QBFRの q_1 から q_2 の内容を q_3 の後にコピーする。

ii) copy q_1, q_2 before q_3 ;

QBFRの q_1 から q_2 の内容を q_3 の前にコピーする。

指定範囲の内容が入りきれない場合 (行間隔は10単位)、入るところまでコピーされる。

3.8 VIP コマンド

<VIP-command> ::= <range-c> | <define-v-c> | <drop-c> |
 <go-c> | <edit-c> | <grant-c> |
 <revoke-c> | <ulist-c> |
 <end-c> |

VIP コマンドは、視野の管理（定義・削除）及び、視野の利用者を管理する為のコマンドである。

A. RANGE コマンド

〔フォーマット〕 `<range-c> ::= range <range-def> { <range-def> } ;`
`<range-def> ::= ( <var-list> , <grel-name> )`
`<var-list> ::= <var> | ( <var> { , <var> } )`

〔機能〕 視野関係の定義式で参照するGCS関係<grel-name>に対する組変数を宣言する。

〔解説〕 参照するGCS関係は、DQP コマンドで宣言したGCS内の関係でなければならない。

B. DEFINE VIEW コマンド

〔フォーマット〕 `<define-v-c> ::= define view <view-rel-name>`
`( <target-list> ) [ where <qual> ] ;`

〔機能〕 視野スキーマ内に視野関係<view-rel-name>を定義する。

〔解説〕 視野名<view-rel-name>は、16桁以内で、頭1文字が英文字の名前を定義する。定義した利用者が、この視野の管理者となる。

C. DROP コマンド

〔フォーマット〕 `<drop-c> ::= drop <view-rel-name>`
`{ , <view-rel-name> } ;`

〔機能〕 視野スキーマ内から指定された視野関係<view-rel-name>を消去する。

〔解説〕 視野スキーマの管理者だけが、視野関係を削除することができる。

D. GO コマンド

〔フォーマット〕 `<go-c> ::= go { <qbf#> } ;`

〔機能〕 作業領域(QBFR)内に定義された視野関係の情報をDI内に格納する。

〔解説〕

i) go :

直前に定義された視野関係をDI内に格納する。

ii) $g o \langle q b f \# \rangle$;

指定された行番号<qbf#>から始まる視野関係の定義式をDI内に格納する。

視野関係の定義式に誤りがあれば、D I 内には格納されない。もし、同一名の視野関係があれば、格納されている視野関係を消去してから D I 内に格納する。

E. GRANT コマンド

```

[フォーマット]  <grant-c> ::= grant <view-schema-name>{, <access-
                                     list>}
                                     to <user-id>{, <user-id>}:
<access-list> ::= all | <access-mode>{, <access-
                                     mode>}
<access-mode> ::= Retrieve | Append | Replace | Delete

```

〔機能〕 利用者<user-id>にDI内に定義されている視野スキーマ<view-schema-name>の利用権を与える。

〔解 説〕 視野スキーマの利用権を与えることができるのは、この視野スキーマを定義した利用者だけである。

<access-list>が省略された場合、規定値としてallがとられ、検索、更新（追加、変更、削除）権が与えられる。

F. REVOKE コマンド

[illegible]

〔機能〕 利用者<user-id>に対して、視野スキーマの利用権をなく奪する。

〔解 説〕 視野スキーマの利用権のはく奪は、視野スキーマを定義した利用者だけが可能である。

G. VIEW USER LIST コマンド

```
[フォーマット] <ulist-c> ::= ulist <view-schema-name>;
```

〔機能〕 視野スキーマ<view-schema-name>の利用権が与えられている利用者の情報を出力する。

〔解 説〕 視野スキーマを定義した利用者だけが使用できる。

H. END コマンド

〔フォーマット〕 `<end-e> :: = end ;`

〔機 能〕 V I P セッションを終了させ、1 レベル上の状態にもどる。

付記3 LDP 使用手引

ここでは、LDP-V3の使用方法について述べる。ここで使用している例題をそのまま実際に実行すれば、同一の結果が得られる。

LDPのインタフェースの詳細については、LDPコマンド〔付記1〕を参照されたい。これは、LDPの全てのコマンドをカバーするものではない。

1. LDPの起動

LDP-V3は、FACOM M-170F(OS IV / F4)のAIMの下で稼動するシステムである。LDP-V3は、TSS端末を通じて利用者と対話を行う。

この為、システムの起動以前に、TSS及びAIMが起動されていないなければならない。TSSが起動されているれば、端末画面に以下のメッセージが表示される。

```
JCB931I PLEASE LOGON.
```

端末を使用する為、以下の様なLOGONコマンドを入力する。

```
LOGON TSS GDP0110/GDP PROC(LDPV3)
```

↑ ↑ ↑
利用者識別名 パスワード ログオンプロシジャー名

ここでPROC(LDPV3)は、LDP-V3の動作環境設定(AIMの使用)を行う。システムからREADYメッセージが表示される。

```
GDP0110 LOGON IN PROGRESS AT 17:11:31 ON FEBRUARY 16, 1983  
NO BROADCAST MESSAGES  
READY
```

次にLDP-V3の起動(EXECコマンド)を行う。

```
EXEC MMUI(LDPV3C)
```

LDPが起動されると、以下の様なメッセージが表示される。

```
*** JDOBS LDP-V3 START... 17.12.08.061
```

続いて、LDPからコマンド入力を促すプロンプトとして#nnnnn(nnnnnは行番号)が表示される。まず最初に、LDPに対して、利用者識別名とパスワードによってLOGONコマンドを実行する。

```
#00010 LOGON LDP YOKOM/YOKMM;
```

↑ ↑
利用者識別名 パスワード

指定した利用者識別名/パスワードが登録されていないと、次のメッセージが表示される。

```
YOUR PASSWORD IS NOT AUTHORIZED  
PLEASE LOGON
```

上記は、パスワードが誤っている場合で、正しいのはYOKOMである。

```
#00020 LOGON LDP YOKOM/YOKOM;  
LDP IS LOGONED
```

LDPを利用する利用者は、あらかじめ管理者に利用者識別名とパスワードの登録をしてもらう必要がある。

LDPがログオンされたら、次に使用するデータベースを指定するQTPコマンドを入力する。
DCDBS (LCS名にLCSDCDBS)を使用する場合、次の様にQTPコマンドを入力する。

```
#00030 QTP LCSDCDBS;  
LCSDCDBS IS OPENED
```

利用者に対して、データベースの利用権が与えられていないと、次の様なメッセージが表示される。

```
YOUR LCS-NAME IS NOT AUTHORIZED
```

この場合、管理者からデータベースの利用権を取得しなければならない。ここでは、管理者が、データベースの利用権を与えるHIPコマンドについては触れない(LDPインタフェースを参照)。データベースの利用者が必要とするQTPコマンドの使い方を説明する。

データベースのLCS名を忘れた場合、次のHELP LCSコマンドを使って、登録されているLCS名を表示させる事ができる。

```
#00040 HELP LCS;  
LCSDCDBS      AT M170-F  
LCSLRSSDB     AT M170-F  
LCSPROJDB     AT M170-F
```

) 登録されているLCS名

次に、問合せに先立って参照する関係に対する組変数を宣言する。

```
#00050 RANGE (D,DOCUMENT);
```

ここでは、関係DOCUMENTに対して組変数Dを宣言した。

組変数は、LDPセッション(LOGONから、LOGOFF)内で有効である。システムの制限として最大32個までの変数を宣言することができる。

例として、図書(DOCUMENT)の受入番号、タイトルを求める問合せを作成する。図書の属性名が不明な場合、次のHELP RELコマンドで関係の属性構成を表示する事が出来る。

#00190 HELP REL DOCUMENT;

ITYPI	REL-NAME	IAT#	IATT-NAME	IES#	ITILE	LENGTH	ITARGE-NAME	IRIMI
IESR	DOCUMENT	1	JNO	01C	8	0	DOCUMENT	1101
IESR	DOCUMENT	2	JND	01I	7	1	DOCUMENT	1010
IESR	DOCUMENT	3	ISEN	01C	20	2	DOCUMENT	1010
IESR	DOCUMENT	4	C-CODE	01I	4	3	DOCUMENT	1010
IESR	DOCUMENT	5	C-DESCRIPT	01C	10	4	DOCUMENT	1010
IESR	DOCUMENT	6	ITYP	01C	1	5	DOCUMENT	1010
IESR	DOCUMENT	7	TITLE	01C	400	6	DOCUMENT	1010
IESR	DOCUMENT	8	NO-OF-AUTHOR	01I	2	7	DOCUMENT	1010
IESR	DOCUMENT	9	P-YEAR	01I	2	8	DOCUMENT	1010
IESR	DOCUMENT	10	P-MONTH	01I	2	9	DOCUMENT	1010
IESR	DOCUMENT	11	P-COMPANY	01C	100	10	DOCUMENT	1010
IESR	DOCUMENT	12	COST	01I	8	11	DOCUMENT	1010
IESR	DOCUMENT	13	PURCH-YEAR	01I	2	12	DOCUMENT	1010
IESR	DOCUMENT	14	PURCH-MONTH	01I	2	13	DOCUMENT	1010
IESR	DOCUMENT	15	PURCH-DAY	01I	2	14	DOCUMENT	1010
IESR	DOCUMENT	16	FISCAL-YEAR	01I	2	15	DOCUMENT	1010
IESR	DOCUMENT	17	ORDER-YEAR	01I	2	16	DOCUMENT	1010
IESR	DOCUMENT	18	ORDER-MONTH	01I	2	17	DOCUMENT	1010
IESR	DOCUMENT	19	ORDER-DAY	01I	2	18	DOCUMENT	1010
IESR	DOCUMENT	20	ORDER-NO	01I	5	19	DOCUMENT	1010
IESR	DOCUMENT	21	CODE-IN-PROJ	01C	10	20	DOCUMENT	1010
IESR	DOCUMENT	22	IDL	01C	6	21	DOCUMENT	1010
IESR	DOCUMENT	23	INUMB	01I	4	22	DOCUMENT	1010
IESR	DOCUMENT	24	TFAGE	01I	5	23	DOCUMENT	1010
IESR	DOCUMENT	25	LANGUAGE	01C	1	24	DOCUMENT	1010

受入番号はJNO、タイトルはTITLEである。RETRIEVE文は、次の様である。

#00200 RETRIEVE INTO R1 (D.JND,D.TITLE);

結果関係名 受入番号 タイトル

実行は、GOコマンドの入力によって開始される。

#00210 GO;

RETRIEVE INTO R1 (JNO=D.JNO , TITLE=D.TITLE);

```

THE ELAPSE TIME FOR RETRVP.QM.. 65
THE ELAPSE TIME FOR GTG..... 16
THE ELAPSE TIME FOR ROGP..... 11
THE ELAPSE TIME FOR DOGP..... 21

THE ELAPSE TIME FOR DFA1..... 3
THE ELAPSE TIME FOR DML..... 1
THE ELAPSE TIME FOR DMLG..... 1221
---ESTIMATED ACCESS OCCURENCE NUMBERS----- OCAT = 0.000---
---ESTIMATED TOTAL SELECTIVITY----- TSCT = 0.10000E+01---
THE ELAPSE TIME FOR DOGD..... 3748

THE ELAPSE TIME FOR SORT..... 6418 KE0562501 JOB GDP0110A SUBMITTED

```

実行した問合せの状態(実行中、正常終了、異常終了)を見る為に、STATUSコマンドを入力する。

#00240 STAT;
R1 IS EXECUTING

上記のメッセージは、問合せ(R1)が実行中であることを示している。

#00250 STAT R1;
R1 IS COMPLETED, WAITING FOR OUTPUT RETRIEVE OCC. =000000279

問合せが正常に終了した時、上記の様なメッセージが表示される。

終了した問合せの結果に対する情報（カーディナリティ格納件数、属性構成）を見るには、HELP RREL コマンドを入力する。結果問合せ名が省略された場合は、利用者の全ての結果問合せの情報が出力される。

#00320 HELP RREL;

(REL-NAME=R1 , DEGREE= 2, CARDINALITY= 279, WIDTH= 151,
TPMODE=0, FLAG=0, EXIST=1, ENCODE=0000)

1	ATT-NAME	1	TYP	1	LN	1	TH	1	CARDIN	1	LT	1	Y	1	ATYP	1	DOMAIN
1	JNO	1	I	1	7	1		1	279	1		1		1	O	1	JNO
2	TITLE	1	C	1	144	1		1	277	1		1		1	O	1	TITLE

結果の内容を表示するOUTPUT コマンドは、次の様である。

#00330 OUTP R1 (JNO/17,TITLE/C50);

** R1

* RESULT CARDINALITY = 000000279

1	JNO	1	TITLE	1
1	712	1	AN INTRODUCTION TO PROBABILITY THEORY AND ITS APPLI	1
1		1	CATIONS, 3RD ED.	1
1	119	1	1 QUEUEING SYSTEMS	1
1	122	1	3 DATA BASE MANAGEMENT	1
1	124	1	7 DATA BASE DESCRIPTION	1
1	138	1	3 QUEUEING SYSTEMS: COMPUTER APPLICATIONS	1
1	139	1	8 LECTURE NOTES IN COMPUTER SCIENCE DATABASE SYSTEMS	1
1	146	1	2 IMODELLING IN DATA BASE MANAGEMENT SYSTEMS	1
1	153	1	8 PROCEEDINGS OF THE INTERNATIONAL CONFERENCE ON VER	1
1		1	Y LARGE DATABASES	1
1	161	1	4 THE ANSI/SPARC DBMS MODEL	1
1	161	1	5 AN INTRODUCTION TO DATABASE SYSTEMS	1
1	162	1	1 DATA BASE ORGANIZATION FOR DATA MANAGEMENT	1
1	162	1	5 ARCHITECTURE AND MODELS IN DATA BASE MANAGEMENT SY	1
1		1	STEMS	1
1	163	1	8 PROCEEDINGS OF THE SECOND BERKELEY WORKSHOP ON DIS	1
1		1	TRIBUTED DATA MANAGEMENT AND COMPUTER NETWORKS	1
1	172	1	7 SYSTEMS FOR LARGE DATA BASES	1
1	178	1	0 DATA TYPES AND STRUCTURES	1
1	178	1	2 PROCEEDINGS OF THE THIRD BERKELEY WORKSHOP ON DIS	1
1		1	TRIBUTED DATA MANAGEMENT AND COMPUTER NETWORKS	1
1	181	1	5 DATA BASE MANAGEMENT SYSTEMS	1
1	184	1	0 INFORMATION SYSTEMS METHODOLOGY LECTURE NOTES IN C	1
1		1	OMPUTER SCIENCE	1
1	186	1	4 DATABASE PROCESSING	1
1	186	1	5 DATABASES: IMPROVING USABILITY AND RESPONSIVENESS	1

結果の出力フォーマットで実際の属性長よりも短く指定した場合、指定桁数分出力し、残りは折り返して出力される。逆に実際の属性長よりも大きく指定した場合、文字タイプでは右側に、数値タイプでは左側に余分な空白を埋める。

結果関係R1を保存しておきたい場合、次の様にMODE CHANGE コマンドで、格納モードをPermanent に変更する。

#00340 MODEC R1 P;
THE MODE OF R1

WAS CHANGED BY P

問合せの比較式の比較演算子としては、次のものが使用出来る。

AND

OR

NOT

EQ 又は = (equal)

NE $\neq$ (not equal)

GT > (greater than)

GE $\leq$ (greater than or equal)

LT < (less than)

LE $\leq$ (less than or equal)

条件式は記述された順に(左から右へ)評価される。

例えば、「図書の受入番号が2200、2300、2400でFiscal-yearが55年の図書を求める」条件式は次の様になる。

#00130 (D.JNO = 2200 OR D.JNO = 2300 OR D.JNO = 2400) AND
#00140 D.FISCAL-YEAR = 55

関係間の主属性間の比較式(主結合式)では、

EQ or = (equal)

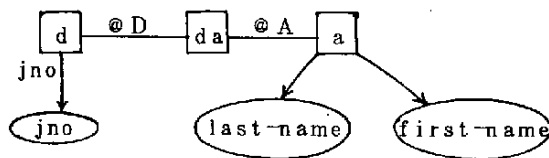
NE or $\neq$ (not equal)

が比較演算子として使用出来る。

例えば、「図書(DOCUMENT)と著者(AUTHOR)の関係から、図書の番号(JNO)とその著者の名前を求める場合」、RETRIEVE文は次の様である。

#00350 RANGE (DA,DOC-AUTH) (A,AUTHOR);
#00360 RETRIEVE INTO R2 (D.JNO,A.LAST-NAME,A.FIRST-NAME)
#00370 WHERE D.DD = DA.DD AND DA.DA = A.DA;

この問合せの問合せグラフは次の様である。



著者名の頭1文字がSより大きい文字(アルファベットのS以降)から始まる著者名の検索は、次の様に記述される。

#00400 RETRIEVE INTO R3 (A.LAST-NAME, A.FIRST-NAME)
#00410 WHERE
#00420 A.LAST-NAME GE "S";
#00430 GO;

実行結果は、次の通りである。

(LAST-NAME	(FIRST-NAME)
ISAGER	IN.
ISALTON	IG.
ISANDERSON	IJ.
ISATA	IT.
ISATTLEY	IJ.
ISAYWARD	IF.
ISCHAEFFER	IJ.
ISCHEIFLER	IR.
ISCHMOLZE	IJ.
ISCHNEIDER	IH.
ISCHNEIDER	IJ.
ISCHNEIDERMAN	IB.
ISCOTT	ID.
ISHAPIRO	IL.
ISHAW	IM.
ISHIPMAN	ID.
ISHNEIDERMAN	IB.
ISHOSHANI	IA.
ISIEGERT	IH.
ISINGH	IM.
ISKWIRZYNSKI	IJ.
ISMITH	IH.
ISOLOMONDIES	IC.
ISLOWLS	IR.
ISPROULL	IB.
ISPRUTH	IW.
ISQUIRE	IM.
ISTEELE	IG.
ISUMMERS	IR.
ISUNSHINE	IC.
ISVOBODOVA	IL.
ITARNLUND	IS.
ITENNANT	IH.
ITENNANT	IR.
ITHOMPSON	IJ.
ITHURBER	IK.
ITDBIAS	IJ.
ITOMEK	II.
ITOU	IJ.
ITSICHRITZIS	ID.
ITURKLE	IS.
ITURN	IR.
ITURNER	ID.
IUHLIG	IR.
IULLMAN	IJ.
IVAN DE RIET	IR.
IVETTER	IM.
IVICK	IC.
IWARMAN	IE.
IWASSERMAN	IA.
IWEGNER	IP.
IWEITZMAN	IC.
IWELDON	IJ.
IWEST	IA.
IWIEDERHOLD	IG.
IWILF	IH.
IWINSTON	IP.
IWOLF	IJ.
IWOLFE	IN.
IWONG	IE.
IWOOD	IC.
IYAO	IS.
IYEH	IR.
IYOVILS	IM.
IZANIOLO	IC.
IZIMMERMAN	IS.

次に、前の問合せを「著者名の頭1桁がSからTまでのものを求めよ」に変更して、検索を行う。
 まず、前の問合せを画面に表示させる為、次のdisplay コマンドを実行する。

#00510 DISP;

LINE	STATEMENT
200	IRETRIEVE INTO R1 (D.JNO,D.TITLE);
330	IOUTP R1 (JNO/I7,TITLE/C50);
360	IRETRIEVE INTO R2 (D.JNO,A.LAST-NAME,A.FIRST-NAME)
370	WHERE D.QD = DA.QD AND DA.QA = A.QA;
400	IRETRIEVE INTO R3 (A.LAST-NAME, A.FIRST-NAME)
410	WHERE
420	A.LAST-NAME GE "S";
460	IOUTP R3 (LAST-NAME/C20,FIRST-NAME/C10);

次に編集したい行番号（この場合は420）をedit コマンドで指定する。

#00520 EDIT 420;

編集行の入力を促すプロンプト“>”と行番号が表示されるので、変更する行イメージを入力する。

>00420 A.LAST-NAME GE "S" AND A.LAST-NAME LE "T";;

編集の中身に“;”を入れる場合、“;”を2つ以上重ねて入力する。

変更された内容を表示すると、次の様である。

LINE	STATEMENT
400	IRETRIEVE INTO R3 (A.LAST-NAME, A.FIRST-NAME)
410	WHERE
420	A.LAST-NAME GE "S" AND A.LAST-NAME LE "T";

実行はG O コマンドで行う。G O コマンドは、最新の問合せ文（RETRIEVE、APPEND、REPLACE、DELETE）を実行する。問合せを指定して実行させるには、次の様に行番号を指定したG O コマンドを実行する。

#00530 GO 400;

R3 WAS ALREADY IN SYSTEM STORAGE

編集した問合せを実行する場合、前の問合せが既に関係作業領域（RWS）に存在している。この場合、以前の結果を消去（ERASE コマンド）するか、結果関係名を変更（EDIT コマンド）する必要がある。結果を消去する場合、次の様に行う。

#00540 ERASE R3;

R3 WAS ERASED

以前の結果を消去したら、あらためてG O コマンドを入力して、問合せを実行する。

#00550 GO 400;

RETRIEVE INTO R3 (LAST-NAME=A.LAST-NAME , FIRST-NAME=
A.FIRST-NAME) WHERE A.LAST-NAME)= "S" AND A.LAST-NAME
<= "T" ;

```

THE ELAPSE TIME FOR RETRVP,QM.. 2069
THE ELAPSE TIME FOR GTG..... 25
THE ELAPSE TIME FOR RQGP..... 15
THE ELAPSE TIME FOR DQGP..... 12
THE ELAPSE TIME FOR DFA1..... 3
THE ELAPSE TIME FOR DML..... 0
THE ELAPSE TIME FOR DMLG..... 1167
---ESTIMATED ACCESS OCCURRENCE NUMBERS----- DCAT = 2.000---
---ESTIMATED TOTAL SELECTIVITY----- TSLCT = 0.25000E+00---
THE ELAPSE TIME FOR DQGD..... 1268

```

実行結果は、次の通りである。

#00580 OUTP R3 (LAST-NAME/C30,FIRST-NAME/C10);

** R3 **

* RESULT CARDINALITY = 000000031

LAST-NAME	FIRST-NAME
ISAGER	IN.
ISALTON	IG.
ISANDERSON	IJ.
ISATA	IT.
ISATTLEY	IJ.
ISAYWARD	IF.
ISCHAFER	IJ.
ISCHEIFLER	IR.
ISCHMOLZE	IJ.
ISCHNEIDER	IH.
ISCHNEIDER	IJ.
ISCHNEIDERMAN	IR.
ISCOTT	IR.
ISHAPIRO	IL.
ISHAW	IN.
ISHIPMAN	ID.
ISHNEIDERMAN	IB.
ISHOSHANI	IA.
ISIEGERT	IH.
ISINGH	IM.
ISKWIRZYNSKI	IJ.
ISMITH	IH.
ISOLOMONDIES	IC.
ISPOWLS	IR.
ISPROULL	IB.
ISPRUTH	IW.
ISQUIRE	IM.
ISTEELE	IG.
ISUMMERS	IR.
ISUNSHINE	IC.
ISVOBODOVA	IL.

集積関数として、min、max、sum、aver、count、anyが使用出来る。例として、「図書館の最も高い価格を求める問合せ」は、次の様に記述される。

#00750 RETRIEVE INTO R6 (MAX-COST=MAX(D.COST));
#00760 GO;

MAX - C O S Tは、問合せの結果属性名を表わし、aggregate 関数の値がセットされる。

```
#00780  OUTP R6 (MAX-COST/I9);
** R6
* RESULT CARDINALITY = 000000001
-----
|MAX-COST |
-----
| 558001
-----
```

「図書の最も安い価格を求める問合せ」は、次の様である。

```
#00790  RETRIEVE INTO R7 (MIN-COST=MIN(D.COST));
#00800  GO;
```

実行結果は、次の通りである。

```
#00850  OUTP R7 (MIN-COST/I9);
** R7
* RESULT CARDINALITY = 000000001
-----
|MIN-COST |
-----
| 01
-----
```

「最も価格の高い図書の受入番号、タイトル、価格を求める問合せ」は、次の様である。

```
#00860  RETRIEVE INTO R8 (D.JNO, D.TITLE, D.COST)
#00870  WHERE
#00880  D.COST = MAX (D.COST);
#00890  GO;
```

問合せの条件式は、求める図書が、価格の最も高い図書である為の条件を表わしている。実行結果は、次の通りである。

```
#00920  OUTP R8 (JNO/I7, TITLE/C50, COST/I9);
** R8
* RESULT CARDINALITY = 000000001
-----
|JNO |TITLE |COST |
-----
| 2315|DISTRIBUTED PROCESSING WITH IBM | 558001
-----
```

「年度毎に、最も価格の高い図書のタイトル、価格、年度を求める」問合せは、group-byを用いて次の様である。

```
#00930  RETRIEVE INTO R10 (D.TITLE, D.COST, D.FISCAL-YEAR)
#00940  WHERE
#00950  D.COST = MAX (D.COST BY D.FISCAL-YEAR);
#00960  GO;
```

実行結果は、次の通りである。

#01070 OUTP R10 (TITLE/C40, COST/I9, FISCAL-YEAR/I2);

** R10 **
* RESULT CARDINALITY = 000000014

ITITLE	ICOST	IFII
IAN INTRODUCTION TO PROBABILITY THEORY AND ID ITS APPLICATIONS, 3RD ED.		01471
IAUTOMATED THEOREM PROVING: A LOGICAL BAS IS	16000	54
IDATA BASE DESCRIPTION	12160	50
IDATA BASEORGANIZATION FOR DATA MANAGEMEN IT	13600	52
IDATA DESIGN: INVITED PAPERS (INFOTECH STAT IE OF THE ART REPORT)	48000	55
IDATA DESIGN: ANALYSIS (INFOTECH STATE OF T IE ART REPORT)	48000	55
IDISTRIBUTED PROCESSING WITH IBM	55800	56
ENCYCLOPEDIA OF COMPUTER SCIENCE AND TEC HNOLOGY	34810	57
INFOTECH STATE OF THE ART REPORT LIFE CY CLE MANAGEMENT: INVITED PAPERS	48000	55
INFOTECH STATE OF THE ART REPORT, LIFE C YCLE MANAGEMENT: ANALYSIS	48000	55
IMODELLING IN DATA BASE MANAGEMENT SYSTEM IS	13500	51
IOFFICE AUTOMATION: ANALYSIS (INFOTECH STAT IE OF THE ART REPORT)	48000	55
IOFFICE AUTOMATION: INVITED PAPERS (INFOTE CH STATE OF THE ART REPORT)	48000	55
ISYSTEMS FOR LARGE DATA BASES	8600	53

「購入図書の平均価格を求める」問合せは、次の様である。

#01190 RETRIEVE INTO R12 (HEIKIN2=AVER(D,COST, D.20));
#01200 GO;

D. @Dは、図書の主キーをユニーク変数として定義する。これは、価格の等しいものを、個々に別のものとして扱う為のものである。

#01270 OUTP R12 (HEIKIN2/I9);

** R12 **
* RESULT CARDINALITY = 000000001

HEIKIN2 I
99631

「著者 (AUTHORR) に登録されている組の数を求める」場合、count を用いて次の様に記述される。

#01290 RETRIEVE INTO R13 (ACNT=COUNT(A.2A));
#01300 GO;

A. @Aは、著者の主キーである。実行結果は、次の通りである。

#01330 OUTP R13 (ACNT/I9);

** R13
* RESULT CARDINALITY = 000000001

ACNT	1
1	2681

数値タイプの属性に対して、算術演算を行う事ができる。算術演算の結果を結果属性とする場合、次の様に結果属性名を定義する。

<result-att>=<expression>

以下の算術演算がサポートされる。

+: 加算

-: 減算

/: 割算

*: 掛算

例えば、図書の属性P-COMPANYの選択度(の10000倍)を求める場合、算術演算と集積関数を用いた問合せは、次の様である。

```
#00630 RETRIEVE INTO R4 (SLCT=COUNT (D.P-COMPANY) * 10000 /  
#00640 COUNT (D.P-COMPANY, D.DD));  
#00650 GQ;
```

実行結果は、次の通りである。

#00680 OUTP R4 (SLCT/I9);

** R4
* RESULT CARDINALITY = 000000001

SLCT	1
1	20431

著者が平均いくつの図書と関係をもつかを示す結合度を求める場合、問合せは次の様である。

```
#00690 RETRIEVE INTO R5 (CDA=COUNT(A.DA, D.DD  
#00700 WHERE A.DA = DA.DA AND DA.DD = D.DD)  
#00710 * 1000 / COUNT (A.DA));  
#00720 GQ;
```

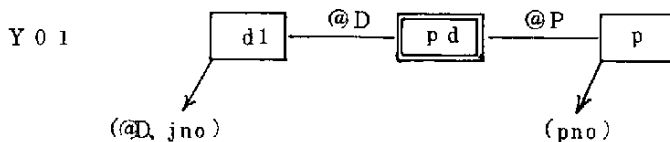
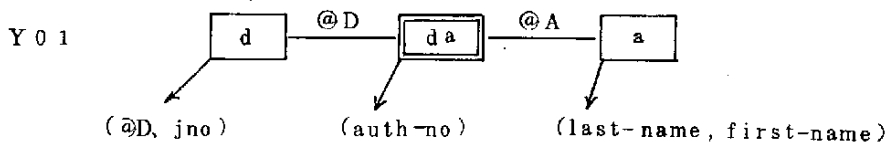
実行結果は、次の通りである。

#00740 OUTP R5 (CDA/I9);

** R5
* RESULT CARDINALITY = 000000001

CDA	1
1	12351

次に視野の定義と視野関係を参照する問合せの例を示す。視野としては、図の様な視野を定義する。



```
#00030 RANGE ((D,D1),DOCUMENT) (DA,DOC-AUTH) (A,AUTHORR);
#00040 RANGE (PD,P-D) (P,PROJECT);
#00050 DEFINE VIEW YOKO
#00060 Y01 (D.@D ,D.JNO, DA.AUTH-NO, LN=A.LAST-NAME, FN=A.FIRST-NAME)
#00070 WHERE
#00080 D.@D = DA.@D AND DA.@A = A.@A;
#00090 Y02 (D1.@D, D1.JNO, P.PNO)
#00100 WHERE
#00110 D1.@D = PD.@D AND PD.@P = P.@P;
```

視野Y 0 1に対して、AUTH-NO (著者番号) が4以上の図書番号と著者名を求める問合せは、次の様である。

```
#00170 RANGE (Y,Y01);
#00180 RETRIEVE INTO VIEW01 (Y.JNO, Y.LN)
#00190 WHERE Y.AUTH-NO GE 4;
```

視野関係Y 0 1を参照する問合せは、問合せ変形によって次の様にLCS関係を参照する問合せに変形される。

```
RETRIEVE INTO VIEW01 ( JNO=D.JNO , LN=A.LAST-NAME ) WHERE
DA.AUTH-NO )= 4 AND D.@D = DA.@D AND DA.@A = A.@A ;
この問合せ結果は次の様である。
```

```
#00300 OUTP VIEW01 (JNO/I7, LN/C30);
```

```
** VIEW01 **
* RESULT CARDINALITY = 000000030
```

JNO	LN
1886	ISQUIRE
2174	SCHMOLZE
2177	BERNSTEIN
2177	GOODMAN
2177	HAMMER
2177	REEVE
2177	ROTHNIE
2177	SHIPMAN
2179	BERNSTEIN

2179	ROTHNIE
2183	MALTZMAN
2184	ANDERSON
2184	BERNSTEIN
2184	GOODMAN
2184	HAMMER
2184	REEVE
2184	ROTHNIE
2186	CARLING
2186	THOMPSON
2187	MALTZMAN
2187	WOLFE
2189	ANDERSON
2189	BERNSTEIN
2189	GOODMAN
2189	HAMMER
2189	REEVE
2189	ROTHNIE
2189	WONG
2214	SOLOMONDIES
2412	KUNII

視野関係 Y 0 1 と Y 0 2 とを @D で結合をとった問合せは、次の様に記述される。

```
#00220 RANGE (Y1,Y01);
#00230 RANGE (Y2,Y02);
#00240 RETRIEVE INTO VIEW02 (Y1,JNO,Y2,PNO)
#00250 WHERE
#00260 Y1.DD = Y2.DD AND Y1.AUTH-NO GE 4;
#00270 GO;
```

この視野問合せは、次の様な L C S 問合せに変形されて実行される。

```
RETRIEVE INTO VIEW02 ( JNO=D.JNO , PNO=P.PNO ) WHERE D.DD =
D1.DD AND DA.AUTH-NO )= 4 AND D1.DD = PD.DD AND
PD.AP = P.AP AND D.DD = DA.DD AND DA.AA = A.AA ;
```

実行結果は、次の様である。

```
#00310 OUTP VIEW02 (JNO/I7,PNO/C8);
```

```
** VIEW02 **
* RESULT CARDINALITY = 000000011
```

IJNO	IPNO
1886	DDBS80
2174	MMUI80
2177	MMUI80
2179	MMUI80
2183	DDBS80
2184	DDBS80
2186	DDBS80
2187	DDBS80
2189	DDBS80
2214	MMUI80
2412	MMUI80

検索問合せで関係の全属性を目標属性とする場合、ALLを使用する事ができる。EMPLOYEE
に対するALLを指定した例を次に示す。

```
#00030  RANGE (E.EMPLOYEE);
#00040  RETRIEVE INTO RREL00 (E.ALL)
#00050  WHERE E.END = 272;
#00060  GO;
```

```
RETRIEVE INTO RREL00 ( ECODE=E.ECODE , FIRST-NAME=E.FIRST-NAME
, LAST-NAME=E.LAST-NAME , DEPT=E.DEPT , EXT=E.EXT , END=
E.END ) WHERE E.END = 272 ;
```

RQLでは、更新コマンドとして次の3つがある。Append、Replace、Delete。更新コマンド
に関しては、PRDBSを更新する例題でその使用方法を示す。

例えば、従業員に、ename="YOKOTSUKA"、efname="MINORU"を追加する場合、appendコマンドは次の様である。

```
#00030  RANGE (E.EMPLOYEE);
#00040  RANGE (O.ORGANIZATOR);
#00050  APPEND TO EMPLOYEE
#00060  (ENAME = "YOKOTSUKA",
#00070  EFNAM = "MINORU")
#00080  WHERE O.ORG-NAME = "JIPDEC";
```

従業員は、17の属性から構成されるが、ここでは2つの属性しか値を設定しない。LCSのHI
登録時にappendで省略可とした属性については、あらかじめ規定された空値(文字タイプには空
白、数値タイプにはゼロ)がセットされる。但し、省略不可(NNA、Null is not allowed)
指定がある属性を追加時に省略する事は出来ない。省略出来ない属性がある場合(例ではenoがNNA
属性)、次のメッセージが表示される。

ENO IS NOT SPECIFIED

また、EMPLOYEEはB型(T)関係ORG-EMPの主キーを主キーとして持つE型関係で
ある。従って、ORG-EMP関係の組(@O、@E)の他方を一意に識別する為の条件式を指定す
る必要がある。例題のO.ORG-NAME="JIPDEC"がその条件式である。次にG型関係
PROJ-EMP-LNKの追加の例を示す。G型関係では、関連づけているE型関係(この場合、
PROJECTとEMPLOYEE)を一意に決める条件が条件式内に規定されなければならない。
従って、appendコマンドは次の様に記述される。

```
#00120  RANGE (PE.PROJ-EMP-LNK);
#00130  APPEND TO PROJ-EMP-LNK
#00140  (@P=P.@P, @E=E.@E,
#00150  STATUS = "ON",SYEAR=82,SMONTH=4,
#00160  EYEAR=0,EMONTH=0,PROPOSITION="SECRETARY",
#00170  ROLE = "MEMBER")
#00180  WHERE P.PNO = "SCRTY" AND E.ENAME = "SUGA";
```

EMPLOYEE

@E	ENAME
(SUGA)	SUGA

PROJECT

@P	PNO
(SCR TY)	SCR TY

PROJ-EMP-LNK

追加→

@E	@P	STATNSS	
(SUGA)	(SCR TY)	ON	

(SUGA) = " SUGA "の@Eの値

(SCR TY) = " SCR TY "の@Pの値

次に、従業員の属性（電話番号、部等）の値を変更する場合、replace コマンドは次の様である。

```
#00320 REPLACE E
#00330 ( TEL = "03-456-2512",
#00340 EXT = 22,
#00350 HIRED-YEAR = 82,
#00360 DEPT = "ADMINISTRATION",
#00370 EPOSITION = "MANAGER" )
#00380 WHERE
#00390 E.ENAME = "HIRAI";
```

replace コマンドの場合、関係名を与える代わりに組変数を与える。組変数は range コマンドで参照する前に宣言しておく。replace コマンドの目標リストで指定された属性 (tel, ext, ... etc) だけが値で置換える。但し、指定されていない属性は変更されない。又、関係内の次の属性は replace コマンドでその値を変更することができない。

(i) 関係の主キー属性

(ii) CALC 項目に対応する属性

replace コマンドで、これらの属性の記述がある場合、次の様なメッセージが表示される。

- (i) #00410 REPLACE E (@E = "XXXX");
@E IS PRIMARY-KEY
- (ii) #00430 REPLACE R (RND="AAA");
RND IS A CALC ITEM

次に、G 型関係 (PROJ-SUBJ) の置換では、関連づけている E 型関係 (PROJECT, SUBJECT) の関係を置換する。例えば、主題 "DBTG DML" と関連するプロジェクトを全

て主題 " CODASYL DML " に関連づける replace コマンドは次の様である。

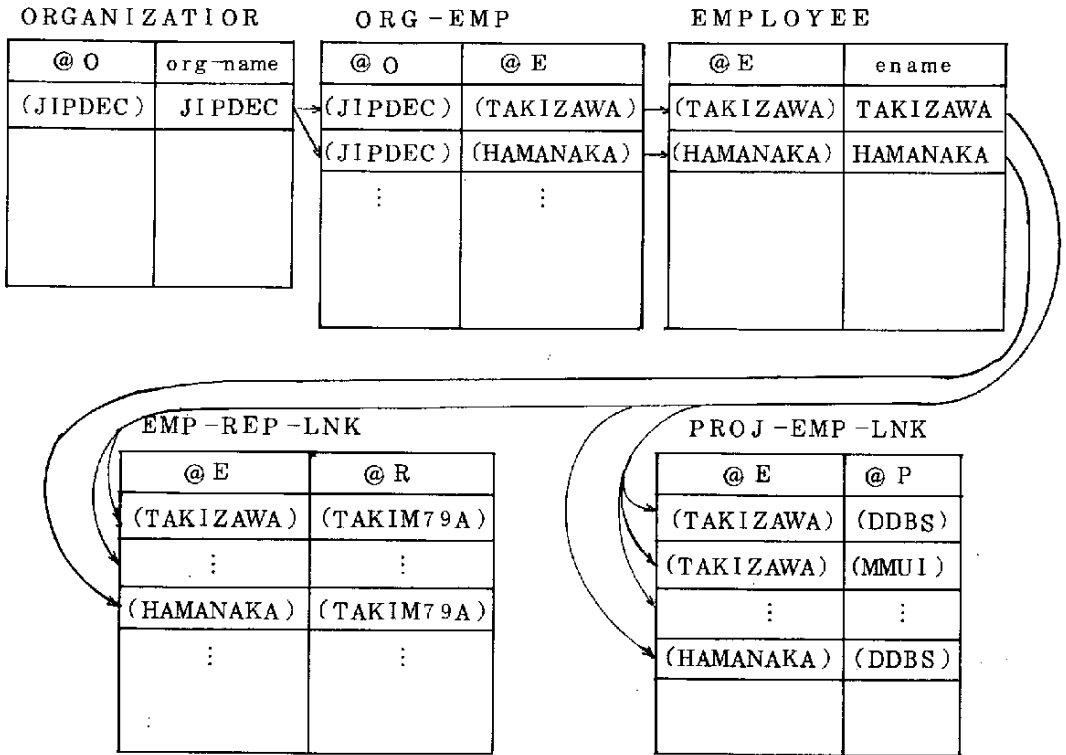
```
#00030 RANGE ((P,P1),PROJECT) ((J,J1),SUBJECT);
#00040 RANGE (PJ,PROJ-SUBJ);
#00050 REPLACE PJ
#00060 ( @P=P.@P, @J=J.@J)
#00070 WHERE
#00080 J1.SUBJECTN = "DBTG DML" AND
#00090 J1.@J = PJ.@J AND PJ.@P = P.@P AND
#00100 J.SUBJECTN = "CODASYL DML";
```

PROJ - SUB J		PROJ - SUB J	
@P	@J		
(MMUI80)	(DBTG DML)	replace →	(MMUI80) (CODASYL...)
(DDBS80)	(DBTG DML)		(DDBS80) (CODASYL...)
:	:		:

次に、ORGANIZATOR(" JIPDEC")を削除する場合、delete コマンドは次の様である。

```
#00110 DELETE O
#00120 WHERE O.ORG-NAME = "JIPDEC";
```

delete コマンドも、関係名でなく組変数名を与える。削除の場合、更新伝播があるので注意を要する。この例の場合、ORGANIZATORの組の削除は、関連づけられているEMPLOYEEの組の削除へ伝播する。又、EMPLOYEEの組の削除は、PROJ-EMP-LNKとEMP-REP-LNKの組の削除へ伝播する。



LDPの利用を終了する為には、まづ終了コマンド (END コマンド) を入力する。

```
#01370 END;
LCSDCDBS IS CLOSED
```

END コマンドが入力されると結果格納領域 (RWS) 内に格納されている temporary モードの結果関係を全て消去する。他のデータベースにアクセスしたい時は、QTP コマンドを入力することによってアクセスできる。

次に、LOGOFF コマンドを入力し、LDP を終了する。

```
#01380 LOGOFF;
***** LOGOFF *****
```

LOGOFF コマンドが入力されると、LDP の終了メッセージと時間が表示される。

```
*** JDDBS LDP-V3 END *** 18.08.43.195
```

付記4 DML I

1. COBOL DML プログラム構造

```
ID-P(10)
  { ID-S(11) { ID-S2(12) } dqg# } gf#
DATA-P(20)
  { DATA-S(21){DATA-S2(22)}dqg#
    { DATA-FR(24)
      { DATA-FI(25)
        { DATA-FA(25-01)} } node# } dqg#
      DATA-AG(26) } gf#
    DATA-WS(30)
      { { DATA-WS1(31)} node# } gf#
        { { { DATA-WS2(32)} node# DATA-WS3(33)DATA-WS4
          (34)} dqg# } gf#
DML-INI(40)
  { { DML-INI1(50-00)
    { DML-INI2(50-01)
      DML-INI3(50-02) } node#
    { DML-P(50-03)} node#
    { DML-TERM(50-10)} node# } dqg#
  { RREL(60) { RREL1(60-01)} dqg# RREL2(60-02)} } gf#
JRP(70)
  { JRP1(70-01)} gf#
  JRP2(70-02)
  UPDT(70-05)
  STOP(70-10)
  { { GNV(80) SRTRN(80-01)} dqg# } gf#
  { { OUTP(82) {OUTP2(82-01)}dqg# OUTP3(82-02)}gf#
FRTRN(81)
  RESULT(81-01)
  LRM      (81-02)
  SUB      (81-03)
```

{ RREL(83)

{ RREL 1(83-01)} dqg#

RREL 2(83-02)} gf #

{ RREL 3、RREL 4(83-01)} dqg#

ERR(84)

UPDT(90)

ここで、

$a(n-m)$: パターン名 a

$n = d \# (\text{division 番号})$

$m = sd \# (\text{division 番号})$

2.DMLの生成

生成される DML statement の形成は次の通りである。

1 2 3 4 5 6 73 74 75 76 77 78 79 80

d# gf# DQG# (COBOL) sd# node# seq#
(DML 文)

sort key 42

- | | | | | |
|---|----|-----------|-----------|-----------------------|
| { | 1) | d # | 昇順 | (Q', Q'' の処理を最後にする) |
| | 2) | g f # | <u>降順</u> | |
| | 3) | D Q G # | 昇順 | |
| | 4) | s d # | " | |
| | 5) | n o d e # | " | |
| | 6) | s e q # | " | |

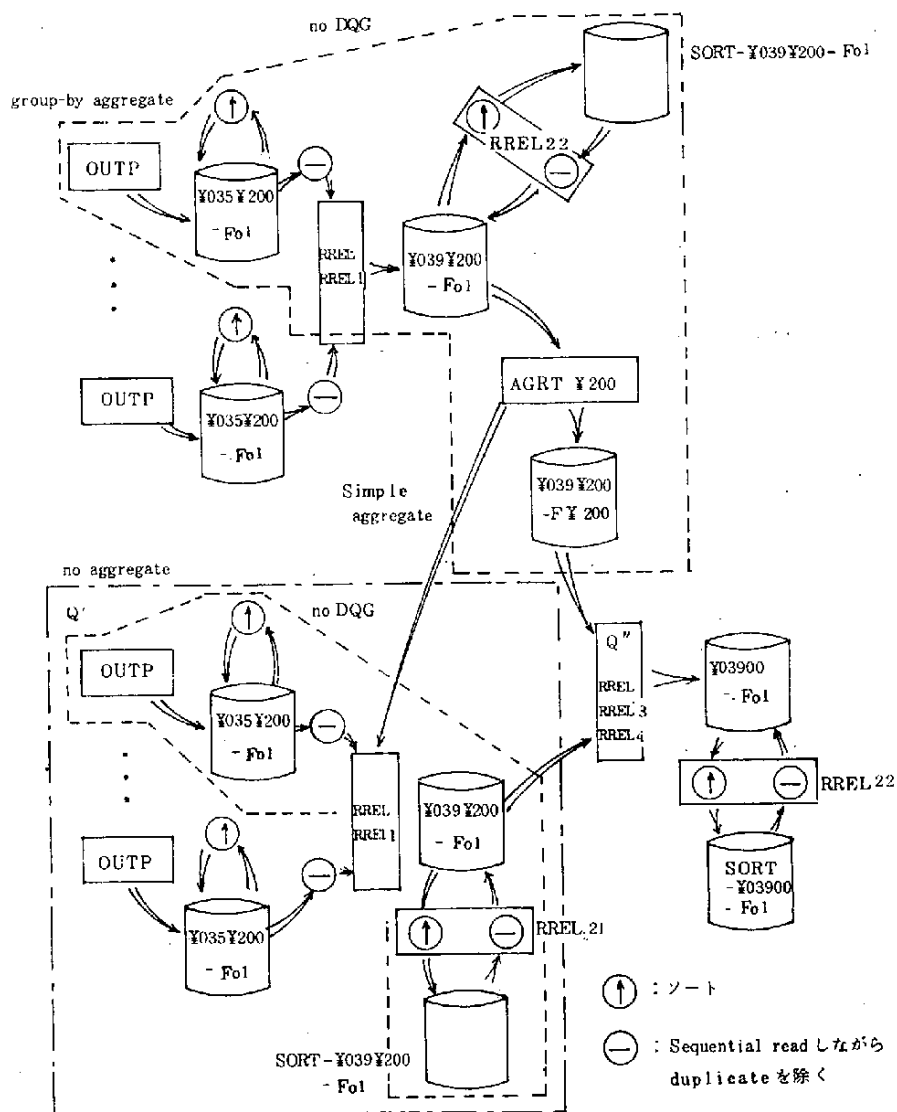


図 2.1 COBOL の中間結果の管理概要

3. パラメータ変換規則とパラメータ内容

DMLパターン内のパラメータの変換規則とパラメータの内容は以下の様である。

[パラメータ変換規則]

i) ¥n

n の値によって、その行を出力するかどうかは異なる。

ii) ¥¥n

変換は¥n と同様に行われるが、その行を出力するかどうかに影響を及ぼさない。

iii) %m

%m は、' LXXXX ' というラベルに変換される。

ノードの
順番 (01 ~) m

%m はその行に ¥n が無い時、その行は出力される。その行に ¥n があれば ¥n に従う。

iv) %%m

%%m は、%m が既に生成されている時のみ出力される。 ☐

表 3.1 % nn

パラメータ	説 明
%00	ATノードのポインタ値
%01	このノードの entry label
%02 ~ %19	paragraph name
%20 ~ %29	内部変数 (%20 は初期値として false がセットされる)
%30 ~ %40	DB-key が格納される変数 (USAGE IS DB-KEY)
%%20 ~ %%29	PROCEDURE DIVISION内で使用された内部変数、又は DB-keyのみ WORKING-STORAGE SECTIONで 定義する。
%%30 ~ %%40	

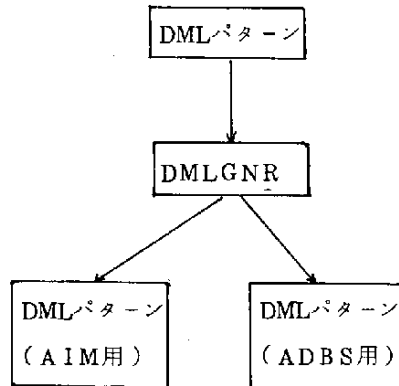
表 3.2 ¥nm1

パラメータ	説 明
¥00	エリア名 (RANGE名)
¥01	レコード型名
¥02	セット型名
¥03	CALC equi-restriction
¥04	¥03以外のequi-restriction
¥05	結果属性のある場合のみ、その行を出力
¥06	false return 先のラベル
¥07	success return 先のラベル
¥08	CALC項目名
¥09	CALC equi-restriction のあるときのみ、その行を出力
¥10	最初の合流ノードのdb-keyが格納されている項目名 (%30)
¥11~¥15	結果属性を順に出力
¥16	P ¥16 が主キー属性のとき、主キー属性名
¥20	tree(P)を文字列に変換する。Pは引数
¥22	operator codeを文字列にする。
¥31	CALCの値
¥32	CALC項目のタイプ
¥33	CALC項目の長さ
¥35	CQGiの結果関係名
¥36	CQGiのnon-supported join formulaの文字列
¥37	CQG#の前ゼロを3ケタで表示する。
¥38	ファイル番号を表示する (file-no=U 1~U 9、V1~V9、W1、W2)
¥39	問合せの結果関係名 (P ¥39)
¥40	First-CN-node SN-node、pro、antiのfirst-nodeの時、その行を出力する。
¥41	2番目以降のCN nodeの時、その行を出力する。
¥42	pro、anti-nodeで2番目以降のnodeの時、その行を出力する。
¥43	last-nodeの時、その行を出力する。
¥45	STB. ac-directionがpositiveの時、その行を出力する。
¥46	STB. ac-directionがnegativeの時、その行を出力する。
¥47	問合せの結果属性名

パラメータ	説 明
¥48	問合せの結果属性のタイプ
¥49	問合せの結果属性のデータ長（前ゼロ3桁で表示）
¥51	¥51→¥11（結果属性に対応する項目）のタイプ
¥52	¥51→¥11（結果属性に対応する項目）の長さ
¥54	次のCQG#（P¥54）
¥55	CQGiの結果関係名（P¥55=CQLのポインタ）
¥56	join formulaの文字列
¥57	結果属性への転送文の生成（P¥57=target-list-treeのポインタ）
¥58	target-listがあるときのみ、その行を出力
¥59	CQGiにjoin formulaがある時のみ、その行を出力する
¥60	failure return先のノードポインタ
¥64	結果属性のタイプ
¥65	結果属性の長さ
¥66	他のノードとの結合式を文字列にする
¥67	RTB#を文字列にする
¥70	success return先のノードポインタ
¥80	disjunctionの数
¥81	アクセス木のノード数
¥83	結果関係の件数（最大値）
¥84	結果属性数
¥85	結果出力リストの見出し
¥86	B-SCHEME（見出し）の長さ
¥87	プログラム名（Lis. os-name）
¥88	AUTHOR名（Lis. la-name）
¥89	host-computer名（Lis. host-computer）
¥90	sub-schema名（Lis. db-name）
¥93	restriction（¥03と¥04）が無い時、その行を出力する
¥94	restriction（¥04）が無い時、その行を出力しない。
¥95	restriction（¥03と¥04）も無く、結果属性（¥05）も無い時、その行を出力しない。
¥96	restriction（¥03）も、結果属性（¥05）も無い時、その行を出力しない。
¥97	スキーマ名

4. DML パターン

DML パターンは、AIM と ADBS の両方をサポートする為に、AIM 用と ADBS 用の DML ステートメントを保持している。実際に、AIM、あるいは ADBS 用の DML 情報は、この DML パターンから各々専用の DML ステートメントだけを持つ DML パターンを生成する。この為、DML パターン内に DML ステートメント区分を設定している。



又、DML ステートメントを分類する時のキーとなる項目 (d #、S - d #、gf #、DQG #、node #) がパターン毎に設定される。

[DML ステートメント区分]

DML パターンの 1 桁目の文字を DML ステートメント区分として使用する。

- ① $\begin{smallmatrix} \downarrow \\ * \\ \downarrow \end{smallmatrix}$ の場合
DML ステートメント内のコメント行を表わす。
- ② $\begin{smallmatrix} \downarrow \\ + \\ \downarrow \end{smallmatrix}$ の場合
AIM 用の DML ステートメント行を表わす。
- ③ $\begin{smallmatrix} \downarrow \\ - \\ \downarrow \end{smallmatrix}$ の場合
ADBS 用の DML ステートメント行を表わす。
- ④ $\begin{smallmatrix} \downarrow \\ (\end{smallmatrix}$, $\begin{smallmatrix} \downarrow \\) \end{smallmatrix}$ の場合
DML ステートメント内の繰返しパターンを表わす。 ☐

パラメータ	説 明
¥100	Sont file のレコード長
¥101	引数
¥200	GF#
¥201	GF#の中間結果関係の属性名 (RATBLより出力)
¥203	aggregate の中間結果領域 (P-V-¥200) の型
¥204	P-V-¥200 のデータ長
¥205	aggregate の目標属性名
¥207	} RATBL内の属性名
¥215	
¥235	Q"の中間結果のGF#
¥239	Q"の中間結果の関係名
¥240	aggregate 関数が count、any の場合、その行を出力しない。
¥241	aggregate の結果属性名
¥256	group-by join式
¥257	Q"の条件式
¥900	Update される要素名 (レコード型名、セット型名、リンクレコード型名) (P¥900は、RRTBLのポインタ)
¥901	現在の主キー属性名
¥902	非主キー属性名 (P¥902は、RATBLの非主キー属性のポインタ)
¥903	非主キー属性名 (ITEM名)
¥904	主キー属性がある時のみ、その行を出力する
¥910	P¥900がセット型のとき、その親レコード型
¥911	P¥900がセット型のとき、その子レコード型
¥915	P¥900がリンクレコード型のとき、現在のレコード型
¥916	P¥900がリンクレコード型のとき、現在のセット型
¥917	P¥900がリンクレコード型のとき、現在のセット型
¥918	Update 関係をメンバとするセット型

```

1 桁目が' ( 'のものの次の行から
   " \ ) ' " 前の行まで
:
(


|                        |
|------------------------|
| * COMMENT              |
| + FIND ANY " ¥ 0 0 1 " |
| -            :         |
| :                      |


)
:

```

この n 行 ($n \geq 1$) が繰り返しパターンとなる。

[DML パターンの分類キー]

各 DML パターンには、以下の様な分類キーが設定される。

d # : division 番号
S - d # : sub - division 番号
gf # : aggregate function 番号
DQG # : DQG 番号
node # : ノード番号

各番号に表われる記号 (*、-、) の意味は次の通りである。

* : 現在の該当番号
- : 0 (ゼロ)
@ : CDQG 番号 □

パターン名	ID - P	d #	s-d #	gf #	DQG #	node #
	DML ブロック	1 0	0	—	—	—
	ID-P 00100000 IDENTIFICATION DIVISION. ***** * ----- ID-P PROGRAM-ID. #087. AUTHOR. #088. * ENVIRONMENT DIVISION. CONFIGURATION SECTION. +SUBSCHEMA-NAME. "090". SOURCE-COMPUTER. #089. OBJECT-COMPUTER. #089. * INPUT-OUTPUT SECTION. FILE-CONTROL. + SELECT PERFM-F ASSIGN TO SETF. + SELECT RELID-F ASSIGN TO RELF.					

パターン名	ID - S	d #	s-d #	gf #	DQG #	node #
	DML ブロック	1 1	0	*	—	—
	ID-S 00110000 ***** + SELECT SORT-#039#200-F01 ASSIGN TO 3 DA-S-SORTWK01 DA-S-SORTWK02 DA-S-SORTWK03.					

パターン名	ID - S01	d #	s-d #	gf #	DQG #	node #
	DML ブロック	1 1	0	*	—	—
	ID-S01 00110000 ***** SELECT #039#200-F01 ASSIGN TO #038.					

パターン名	ID - S2	d #	s-d #	gf #	DQG #	node #
	DML ブロック	1 2	0	*	*	—
	ID-S2 00120000 ***** + SELECT SURT-#035#200-F01 ASSIGN TO 3 + DA-S-SORTWK01 + DA-S-SORTWK02 + DA-S-SORTWK03. SELECT #035#200-F01 ASSIGN TO #038.					

パターン名	ID - S3	d #	s-d #	gf #	DQG #	node #
	DML ブロック	1 3	0	*	—	—
	ID-S3 00130000 ***** SELECT #039#200-F#200 ASSIGN TO #038.					

パターン名	DATA - P	d #	s-d #	gf #	DQG #	node #
	DML ブロック	2 0	0	—	—	—
	DATA-P 00200000 *-----* * DATA-P DATA DIVISION. -SUB-SCHEMA SECTION. -DB #090 WITHIN #097. * FILE SECTION. +FD PERFM-F LABEL RECORD IS STANDARD. +01 PERFM-R. + 03 FILLER PIC X(52). + COPY RESLTID.					

パターン名	DATA - S	d #	s-d #	gf #	DQG #	node #
	DML ブロック	2 1	0	*	—	—
	DATA-S 00210000 *-----* * DATA-S SURT FILE FOR #039#200 +SD SORT-#039#200-F01. +01 SORT-#039#200-R01. + 03 SORT-ITEM-#039#200 PIC X(#100).					

パターン名	DATA - S1	d #	s-d #	gf #	DQG #	node #
	DML ブロック	2 2	0	*	*	—
	DATA-S1 00220000 *-----* * DATA-S1 +SD SORT-#035#200-F01. +01 SORT-#035#200-R01. 03 SORT-ITEM-#035#200 PIC X(#101).					

パターン名	DATA - FR	d #	s-d #	gf #	DQG #	node #
	DML ブロック	2 4	0	*	—	—
	DATA-FR 00240000 *-----* * DATA-FR (RESULT RELATION) FD #039#200-F01 LABEL RECORD IS STANDARD. 01 #039#200-R01. (03 #047 PIC #048(#049).)					

パターン名	DATA - FI	d #	s-d #	gf #	DQG #	node #
	DML ブロック	2 5	0	*	*	—
	DATA-FI 00250000 *-----* * DATA-FI (RESULT RELATION) FD ¥035¥200-F01 LABEL RECORD IS STANDARD. 01 ¥035¥200-R01.					

パターン名	DATA - FA	d #	s-d #	gf #	DQG #	node #
	DML ブロック	2 5	1	*	*	*
	DATA-FA 00250001 * DATA-FA (TARGET ATTRIBUTE) + 03 ¥016¥00 PIC X(08). (03 ¥011¥00 PIC ¥051(¥052).)					

パターン名	DATA - AG	d #	s-d #	gf #	DQG #	node #
	DML ブロック	2 4	1	*	—	—
	DATA-AG 00240001 *-----* * DATA-AG FD ¥039¥200-F¥200 LABEL RECORD IS STANDARD. 01 ¥039¥200-R¥200. (03 ¥215 PIC ¥216(¥217).)					

パターン名	DATA - WS	d #	s-d #	gf #	DQC #	node #
	DML ブロック	3 0	0	—	—	—
	<pre> DATA-WS 00300000 ----- * DATA-WS WORKING-STORAGE SECTION. * 01 NDTBL. 02 NDTBLU OCCURS 1081 TIMES.* 03 NPT PIC 9(05). 03 ND-ND PIC 9(05). 03 LPT-NU PIC 9(05). 03 RPT-ND PIC 9(05). 03 SMLPTNU PIC 9(05). 03 SRPPTNU PIC 9(05). 01 CNDT PIC 9(05) VALUE 0. 01 MXNDT PIC 9(05) VALUE 1081.9 * * SCHEME OF THE RESULT RELATION 01 SRSLT. 02 SRSLTU OCCURS 1084 TIMES.* 03 NPT PIC 9(05). 03 ND-ND PIC 9(05). 03 CVPT PIC 9(05). 03 LENGTT PIC 9(05). 01 CSRT PIC 9(05) VALUE 0. 01 MXSRT PIC 9(05) VALUE 1084.9 01 CSR PIC 9(05) VALUE 1. * * TABLE FOR STORING DBKEYS OF SATISFIABLE OCCURRENCES. 01 STDBK. 02 STDBKEY1 OCCURS 1084 TIMES.* 03 STDBKEY2 OCCURS 1083 TIMES.* 04 DBKEY USAGE IS DB-KEY. 04 DBKEY PIC X(98). * * * STATUS VEHICLE FROM PRECEDING NODE 01 LFOUND PIC 9(02). * * SAVED RECORD (*039-F) AREA 01 WK*039 PIC X(1000). 01 S*039 PIC X(1024). 01 UDBK PIC X(08). * * THE NUMB. OF ACCESSED OCCURRENCES 01 CNT PIC 9(08) VALUE 0. 01 GCNT PIC 9(08) VALUE 0. 01 GCNTD PIC 9(08) VALUE 0. 01 C1 PIC 9(08) VALUE 0. 01 G1 PIC 9(08) VALUE 0. 01 GU1 PIC 9(08) VALUE 0. * * * VARIABLES FOR STORING ELAPSE TIME. 01 ONE PIC 9 VALUE 1. 01 T1 PIC 9(10) VALUE 0. 01 T2 PIC 9(10) VALUE 0. 01 T3 PIC 9(10) VALUE 0. 01 T4 PIC 9(10) VALUE 0. 01 T5 PIC 9(10) VALUE 0. 01 T6 PIC 9(10) VALUE 0. 01 T7 PIC 9(10) VALUE 0. 01 T12 PIC 9(10) VALUE 0. 01 T13 PIC 9(10) VALUE 0. 01 T15 PIC 9(10) VALUE 0. 01 T16 PIC 9(10) VALUE 0. 01 T20 PIC 9(10) VALUE 0. 01 T31 PIC 9(10) VALUE 0. 01 T51 PIC 9(10) VALUE 0. * * VARIABLE FOR STORING THE ELAPSE TIME OF UPDATING. 01 TD0 PIC 9(10) VALUE 0. 01 TD1 PIC 9(10) VALUE 0. </pre>					

パターン名	DATA - WS	d #	s-d #	gf #	DQG #	node #
	DML ブロック	3 0	0	-	-	-
	<pre> 01 TIME1 PIC 9(08). 01 TIME2 REDEFINES TIME1. 03 TM PIC 9(02) OCCURS 4. 01 CTIME PIC 9(10). * * CONSTANT 01 FALSE PIC 9(02) VALUE 0. 01 TRUE PIC 9(02) VALUE 1. 01 ENDD PIC 9(02) VALUE 0. 01 ENDF PIC 9(02) VALUE 0. 01 YES PIC 9(02) VALUE 1. 01 NOO PIC 9(02) VALUE 0. 01 ONN PIC 9(02) VALUE 1. 01 OFFF PIC 9(02) VALUE 0. * * * WORKING VARIABLES 01 I PIC 9(08). 01 J PIC 9(08). 01 O PIC 9(08). 01 OI PIC 9(08). 01 OJ PIC 9(08). 01 H PIC 9(08). 01 K PIC 9(08). 01 L PIC 9(08). 01 M PIC 9(08). 01 N PIC 9(08). 01 II PIC 9(08). 01 JJ PIC 9(08). 01 JH PIC 9(08). 01 KK PIC 9(08). 01 PT PIC 9(08) VALUE 0. 01 PPT PIC 9(08) VALUE 0. 01 PTND PIC 9(08) VALUE 0. 01 PPTND PIC 9(08) VALUE 0. 01 LPTND PIC 9(08) VALUE 0. 01 NRPPTND PIC 9(08) VALUE 0. 01 NLPTND PIC 9(08) VALUE 0. 01 RPPT PIC 9(08) VALUE 0. 01 RPTND PIC 9(08) VALUE 0. 01 NPTND PIC 9(08) VALUE 0. 01 PTN PIC 9(08) VALUE 0. 01 LPT PIC 9(08) VALUE 0. -01 DBK USAGE IS DB-KEY. +01 DBK PIC X(08). +01 DSU PIC X(08). +01 DRN PIC X(08). * * NAME OF RECORD-TYPE (REN) AND ITEM (ITN) 01 REN PIC X(16). 01 ITN PIC X(16). 01 ERH1 PIC 9(03). 01 ERH2 PIC 9(02). 01 WST PIC X(02). 01 CTOTAL PIC 9(08) VALUE 0. 01 CNTDQG PIC 9(08). +01 SETX PIC 9(03) VALUE 13. 01 MODES PIC 9. 01 MODEE PIC 9(05). </pre>					

パターン名	DATA - WS 1	d #	s-d #	gf #	DQG #	node #
	DML ブロック	3 1	0	*	*	*
	DATA-WS100310000 *----- * DATA-WS1 ***** DATA-WS1 (Y037) (%00)e 01 %%20 PIC 9(08).e 01 %%21 PIC 9(08).e 01 %%22 PIC 9(08).e 01 %%23 PIC 9(08).e 01 %%24 PIC 9(08).e 01 %%25 PIC 9(08).e 01 %%26 PIC 9(08).e 01 %%27 PIC 9(08).e 01 %%28 PIC 9(08).e 01 %%29 PIC 9(08).e * USAGE IS DBKEY. +01 %%30 PIC X(08).e +01 %%31 PIC X(08).e +01 %%32 PIC X(08).e +01 %%33 PIC X(08).e +01 %%34 PIC X(08).e +01 %%35 PIC X(08).e +01 %%36 PIC X(08).e +01 %%37 PIC X(08).e +01 %%38 PIC X(08).e +01 %%39 PIC X(08).e +01 %%40 PIC X(08).e -01 %%30 USAGE IS DB-KEY. e -01 %%31 USAGE IS DB-KEY. e -01 %%32 USAGE IS DB-KEY. e -01 %%33 USAGE IS DB-KEY. e -01 %%34 USAGE IS DB-KEY. e -01 %%35 USAGE IS DB-KEY. e -01 %%36 USAGE IS DB-KEY. e -01 %%37 USAGE IS DB-KEY. e -01 %%38 USAGE IS DB-KEY. e -01 %%39 USAGE IS DB-KEY. e -01 %%40 USAGE IS DB-KEY. e * AREA FOR STORING NON-PRIMARY JOIN ATT. (R#067Y#200)e 01 R#067Y#200. (02 #063 PIC #064(#063).e)					

パターン名	DATA - WS 2	d #	s-d #	gf #	DQG #	node #
	DML ブロック	3 2	0	*	*	-
	DATA-WS200320000 *----- * DATA-WS2 ***** DATA-WS2 (Y037Y#200-%00)e * TABLE FOR STORING CALC VALUES. 01 CV0Y037Y#200. 02 CV1Y037Y#200 OCCURS #080 TIMES.e 03 CVY037Y#200 PIC #032(Y033).e 01 MXCVY037Y#200 PIC 9(05) VALUE Y080.e 01 CCVY037Y#200 PIC 9(05) VALUE 0.e 01 CVALY037Y#200 PIC Y032(Y033).e					

パターン名	DATA - WS 3	d #	s-d #	gf #	DQG #	node #
	DML ブロック	3 3	0	*	*	-
	DATA-WS300330000 *----- * CARDINALITY OF ¥035¥¥200. 01 C¥037¥¥200 PIC 9(08) VALUE 0. -01 WST¥037¥¥200 PIC X(02). 01 WK¥035¥¥200 PIC X(1024). 01 S¥035¥¥200 PIC X(1024). 01 M¥035¥¥200 PIC 9.					

パターン名	DATA - WS 4	d #	s-d #	gf #	DQG #	node #
	DML ブロック	3 4	0	*	-	-
	DATA-WS400340000 *----- *01 P-V-¥200 PIC ¥203(¥204). 01 P-V-¥200 PIC S9(09)¥999.					

パターン名	DATA - WS 5	d #	s-d #	gf #	DQG #	node #
	DML ブロック	3 4	0	*	-	-
	DATA-WS500340000 *----- 01 C¥039¥200 PIC S9(09). 01 WK¥039¥200 PIC X(1024). 01 S¥039¥200 PIC X(1024). 01 M¥039¥200 PIC 9. 01 E¥039¥200 PIC 9. 01 N¥039¥200 PIC 9. 01 V¥039¥200 PIC S9(09) VALUE ¥241.					

パターン名	DATA - WS 6	d #	s-d #	gf #	DQG #	node #
	DML ブロック	3 4	0	*	-	-
	DATA-WS600340000 *----- 01 AG-R-¥200. (03 ¥212 PIC ¥213(¥214).)					

パターン名	DML - INI	d #	s-d#	gf #	DQG#	node #
	DML ブロック	4 0	0	—	—	—
	<pre> DML-INI 00400000 ----- * DML-INI ***** DML-INIT PROCEDURE DIVISION. -MAIN SECTION. DML-M. PERFORM GETTIME. MOVE CTIME TO TO. + READY. ***** TRANSACTION START.***** - READY USAGE-MODE IS UPDATE. * INITIALIZATION OF VARIABLES. MOVE 1 TO TRUE ONN YES ENDD. MOVE 0 TO FALSE OFFF NOU. * INITIALIZATION OF SRSLT. MOVE #084 TO MXSRT. * SET ND-NO IN NDTBL. MOVE 1 TO 1. INI2. IF 1 NOT > MXNDT MOVE 1 TO ND-NO IN NDTBL (1) ADD 1 TO 1 GO TO INI2. INI3. MOVE 0 TO T13 T15. MOVE 0 TO CNT GCNT GCNTU. * INITIALIZATION OF SET-PERFORMANCE DATA TABLE Y401 + MOVE 1 TO 1. MOVE 0 TO MAX-SET. Y401 *INIT-SET, Y401 + IF 1 NOT > #400 Y401 + MOVE SPACE TO SET-NAME (1) SET-COMND (1) #401 + MOVE 0 TO SET-MC (1) SET-OC (1) Y401 + SET-IMC (1) SET-IOC (1) #401 + ADD 1 TO 1 GO TO INIT-SET. Y401 </pre>					

パターン名	DML - INI 0	d #	s-d#	gf #	DQG#	node #
	DML ブロック	5 0	0	*	—	—
	<pre> DML-INI000500000 ----- * DML-INI0 MOVE 0 TO C#039#200. IF NOT (#300) OPEN OUTPUT #039#200-F01 CLUSE #039#200-F01 GO TO PRELP#200. </pre>					

パターン名	DML - INI 1	d #	s-d#	gf #	DQG#	node #
	DML ブロック	5 0	1	*	*	—
	<pre> DML-INI100500001 ----- * DML-INI1 ***** OPEN INTERMEDIATE RESULTS(DQG#*037*#200)* OPEN OUTPUT #035#200-F01. PERFORM GETTIME. MOVE CTIME TO T2. * MOVE 0 TO CSRT CNDT. MOVE 1 TO CSR. MOVE FALSE TO IFOUND. </pre>					

パターン名	DML - INI 2	d #	s-d #	gf #	DQG #	node #
	DML ブロック	5 0	2	*	*	*
	DML-INI200500002 *----- * DML-INI2 * INITIALIZATION OF AT-NODES(¥037¥¥200-¥00).@ MOVE FALSE TO ¥20.@ ADD 1 TO CNDT. MOVE ¥00 TO NPT IN NDTBL (CNDT).@ MOVE 0 TO LPT-NO IN NDTBL (CNDT) RPT-NO IN NDTBL (CNDT). ADD 1 TO CSRT. ¥005@ MOVE ND-NU IN NDTBL (CNDT) TO ¥005@ ND-NU IN SRSLT (CSRT). ¥005@ MOVE ¥00 TO NPT IN SRSLT (CSRT).¥005@ MOVE 0 TO CVPT IN SRSLT (CSRT) ¥005 LENGTH IN SRSLT (CSRT).¥005					

パターン名	DATA - INI 3	d #	s-d #	gf #	DQG #	node #
	DML ブロック	5 0	2	*	*	*
	DATA-INI300500002 *----- * DML-INI3 * SET CALC VALUES (¥037¥¥200-¥00).@ MOVE 0 TO CCV¥037¥¥200. (ADD 1 TO CCV¥¥037¥¥200.@ MOVE ¥031 TO CV¥¥037¥¥200 (CCV¥¥037¥¥200).@)					

パターン名	OWNER	d #	s-d#	gf#	DQG#	node#
	DML ブロック	5 0	1 0	*	*	*
	<pre> OWNER 00500010 #-----# # OWNER ##### *** OWNER (*037*200-*00) (*001 VIA *002)@ %01. MOVE %00 TO PT.@ + MOVE "FIND" TO WCDM. + MOVE "*002" TO SET-NAME-W. PERFORM SET-SEARCH. + ADD 1 TO SET-IMC (SET-P). - FIND OWNER WITHIN *002.@ + FIND OWNER WITHIN "*002".@ + IF DB-EXCEPTION IS 12 GO TO %02.@ - IF DB-STATUS = "0503100" GO TO %02.@ + ADD 1 TO SET-IDC (SET-P). PERFORM COUNTR. GO TO %03.@ * NOT FOUND %02. MOVE FALSE TO LFOUND.@ * FAILURE RETURN MOVE *060 TO PPT. PERFORM FRTRN.@ GO TO *006.@ * ANY OCCURRENCE IS FOUND@ %03.@ - ACCEPT %30 FROM CURRENCY.@ + MOVE PGCS OF FCOM TO %30.@ * PRO-OR ANTI-NODE *042@ IF %30 NOT = *010 GO TO %04. GO TO %02.*042@ IF %30 = *010 GO TO %04. GO TO %02.@ %04.@ - GET *001. *095@ + GET CURRENT. *095@ PERFORM GCOUNT. *095@ IF (*003 AND *004) NEXT SENTENCE ELSE GO TO %02.@ (MOVE *061 *075 TO *062 IN R*067*200.@) IF *066 NEXT SENTENCE ELSE GO TO %02.@ MOVE %30 TO DBX. PERFORM RESULT. *005@ * LAST NODE *043@ MOVE TRUE TO LFOUND. *043@ MOVE *070 TO PPT. PERFORM SRTRN*037*200. *043@ GO TO *007. *043@ </pre>					

パターン名	SORT - EQ - DNA	d #	s-d #	gf #	DQC #	node #
	DML ブロック	5 0	1 0	*	*	*
	<pre> SRTEQDNA00500010 #----- # SORT-EQ-DNA # SORT-EQ-DNA(NOT LAST-NEXT-NODE) *** SORT-EQ-DNA(#037**200-#00) (#001 VIA #002)@ %01. MOVE %00 TO PT.@ - IF %20 = TRUE GO TO %06.@ - MOVE TRUE TO %20.@ - FIND FIRST #001 WITHIN #002.@ - GO TO %07.@ -%06.@ + FIND NEXT #001 WITHIN #002.@ + FIND NEXT "#001" WITHIN "#002".@ + IF DB-EXCEPTION IS 11 GO TO %02.@ + IF DB-EXCEPTION IS 12 GO TO %02.@ -%07. IF DB-STATUS = "0502100" GO TO %02.@ - IF DB-STATUS = "0503100" GO TO %02.@ - PERFORM COUNTR. GO TO %03.@ - FAILURE RETURN %02. MOVE FALSE TO LFOUND.@ - MOVE FALSE TO %20.@ - MOVE #060 TO PPT. PERFORM FRTRN.@ - GO TO #006.@ # ANY OCCURRENCE IS FOUND. %03.@ - ACCEPT %30 FROM CURRENCY.@ + MOVE PGCS OF FCUM TO %30.@ # PRQ-OR ANI-NODE. - IF %30 NOT = #010 GO TO %01. #042@ - PERFORM GCOUNT. #095@ + GET #001. #095@ + GET CURRENT. #095@ - PERFORM GCOUNT. #095@ - IF #023 #025 #024 GO TO %02.@ - IF #020 NEXT SENTENCE ELSE GO TO %01.@ - IF #021 NEXT SENTENCE ELSE GO TO %01.@ (MOVE #061 **075 TO #062 IN R**067**200.@) IF #066 NEXT SENTENCE ELSE GO TO %01.@ # %05. MOVE TRUE TO LFOUND.@ - MOVE %30 TO DBK. PERFORM RESULT. #005@ # LAST NODE #043@ - MOVE #070 TO PPT. PERFORM SRTRN#037**200.#043@ - GO TO #007. #043@ </pre>					

パターン名	EQ - DNA	d #	s-d #	gf #	DQG #	node #
	DML ブロック	5 0	1 0	*	*	*
	<pre> EQ-DNA 00500010 ----- * EQ-DNA * EQ-DNA (NOT LAST-NEXT-NODE) *** EQ-DNA (¥037¥¥200-¥00) (¥001 VIA ¥002)* %01. MOVE %00 TO PT.@ - IF %20 = TRUE GO TO %06.@ - MOVE TRUE TO %20.@ - FIND FIRST ¥001 WITHIN ¥002.@ - GO TO %07.@ -%06.@ - FIND NEXT ¥001 WITHIN ¥002.@ + FIND NEXT "¥001" WITHIN "%02".@ + IF DB-EXCEPTION IS 11 GO TO %02.@ + IF DB-EXCEPTION IS 12 GO TO %02.@ -%07. IF DB-STATUS = "0502100" GO TO %02.@ - IF DB-STATUS = "0503100" GO TO %02.@ - PERFORM COUNTR. * FAILURE RETURN %02. MOVE FALSE TO LFOUND.@ - MOVE %060 TO PPT. PERFORM FRTRN.@ - GO TO ¥006.@ * FOUND %03.@ - MOVE FALSE TO %20.@ - ACCEPT %30 FROM CURRENCY.@ + MOVE PGCS OF FCDM TO %30. ¥095@ - PERFORM GCOUNT. ¥095@ + GET ¥001. ¥095@ + GET CURRENT. ¥095@ - IF %20 NEXT SENTENCE ELSE GO TO %01.@ - IF %21 NEXT SENTENCE ELSE GO TO %01.@ (MOVE ¥061 ¥¥075 TO ¥062 IN R¥¥067¥¥200.@) - IF ¥66 NEXT SENTENCE ELSE GO TO %01.@ - MOVE %30 TO DBK. ¥005@ - PERFORM RESULT. ¥005@ * LAST NODE ¥043@ - MOVE TRUE TO LFOUND. ¥043@ - MOVE ¥070 TO PPT. PERFORM SRTRN¥037¥¥200.¥043@ - GO TO ¥007. ¥043@ </pre>					

パターン名	SORT - EQ - DA	d #	s-d #	gl #	DQG #	node #
	DML ブロック	5 0	1 0	*	*	*
	<pre> SORTE00A00500010 ----- **** SORT-EQ-DA *** SORT-LQ-DA (#037*#200-%00) (#001 VIA #002)* %01. MOVE %00 TO PT.* IF %20 = TRUE GO TO %02.* MOVE FALSE TO LFOUND. MOVE TRUE TO %20. MOVE FALSE TO %21.* - FIND FIRST %01 WITHIN %02.* - GO TO %07.* + GO TO %03.* %02. IF LFOUND = TRUE MOVE TRUE TO %21.* - FIND %001; DB-KEY IS %30. #040* + MOVE %30 TO PGCS OF FCUM. #040* + FIND CURRENT. #040* %03.* - FIND NEXT %001 WITHIN %02.* + FIND NEXT "%001" WITHIN "%002".* + IF DB-EXCEPTION IS 11 GO TO %04.* + IF DB-EXCEPTION IS 12 GO TO %04.* -%07. IF DB-STATUS = "0502100" GO TO %04.* - IF DB-STATUS = "0503100" GO TO %04.* PERFORM COUNTR. GO TO %06.* * END %04. MOVE FALSE TO %20. MOVE %21 TO LFOUND.* IF %21 = TRUE GO TO %05.* * FAILURE RETURN MOVE %060 TO PPT. PERFORM FRTN.* GO TO %006.* * SUCCESS RETURN %05. MOVE %070 TO PPT. PERFORM SRTN#037*#200.* GO TO %007.* * ANY OCCURRENCE IS FOUND. %06.* - ACCEPT %30 FROM CURRENCY.* + MOVE PGCS OF FCUM TO %30.* * PRD-OR ANTI-NODE #042* IF %30 = %010 GO TO %03. #042* PERFORM GCOUNT. #095* - GET #001. #095* + GET CURRENT. #095* IF %023 %025 %024 GO TO %04.* IF %020 NEXT SENTENCE ELSE GO TO %03.* IF %021 NEXT SENTENCE ELSE GO TO %03.* * SATISFIABLE OCCURRENCE IS OBTAINED. (MOVE #061 *#075 TO #062 IN R*#067*#200.*) IF %066 NEXT SENTENCE ELSE GO TO %03.* MOVE TRUE TO %21. #043* MOVE %30 TO DBK. #005* PERFORM RESULT. #005* GO TO %03. #043* </pre>					

パターン名	FIRST	d #	s-d #	g f #	DQC #	node #
	DML ブロック	5 0	1 0	*	*	*
	<pre> FIRST 00500010 ----- ***** FIRST *** FIRST (SEQUENTIAL ACCESS WITHIN AREA) * (*001) (*037**200-%00) - FIND FIRST %001 WITHIN %000.@ + FIND FIRST "%001" WITHIN "%000" RANGE.@ + IF DB-EXCEPTION IS 11 GO TO TERM%037**200.@ + IF DB-EXCEPTION IS 12 GO TO TERM%037**200.@ - IF DB-STATUS = "0502100" GO TO TERM%037**200.@ - IF DB-STATUS = "0503100" GO TO TERM%037**200.@ + GO TO %03.@ - GO TO %07.@ %01.@ - FIND %001 ; DB-KEY IS %30.@ + MOVE %30 TO PGCS OF FCDM.@ + FIND CURRENT. %02.@ - FIND NEXT %001 WITHIN %000.@ + FIND NEXT "%001" WITHIN "%000" RANGE.@ + IF DB-EXCEPTION IS 11 GO TO TERM%037**200.@ + IF DB-EXCEPTION IS 12 GO TO TERM%037**200.@ -%07. IF DB-STATUS = "0502100" GO TO TERM%037**200.@ - IF DB-STATUS = "0503100" GO TO TERM%037**200.@ %03. PERFORM COUNTR.@ MOVE %00 TO PT.@ GET %001. %095@ + GET CURRENT. %095@ PERFORM GCOUNT. %095@ IF (%003 AND %004) NEXT SENTENCE ELSE GO TO %02.@ * SATISFIED (MOVE %061 **075 TO %062 IN RY%067**200.@) IF %066 NEXT SENTENCE ELSE GO TO %02.@ - ACCEPT %30 FROM %001 CURRENCY.@ + MOVE PGCS OF FCDM TO %30.@ + MOVE %30 TO DBK. PERFORM RESULT. %005@ * LAST NODE %043@ MOVE %070 TO PPT. PERFORM SRTRN%037**200. %043@ GO TO %02. %043@ </pre>					

パターン名	SORT - THETA	d #	s-d #	gf #	DQG #	node #
	DML ブロック	50	10	*	*	*
	<pre> SRTTHETA00500010 ----- ***** SORT-THETA *** SORT-THETA (#037#200-#00)(#001 VIA #002)@ %01. MOVE #00 TO PT.@ IF #20 = TRUE GO TO #02.@ MOVE FALSE TO LFOUND. MOVE TRUE TO #20. MOVE FALSE TO #21.@ - FIND FIRST #001 WITHIN #002.@ - GO TO #07.@ + GO TO #03.@ %02. IF LFOUND = TRUE MOVE TRUE TO #21.@ - FIND #001; DB-KEY IS #30. #040@ + MOVE #30 TO PGCS OF FCOM. #040@ + FIND CURRENT. #040@ %03.@ - FIND PRIOR #001 WITHIN #002. #046@ + FIND PRIOR "#001" WITHIN "#002". #046@ - FIND NEXT #001 WITHIN #002. #045@ + FIND NEXT "#001" WITHIN "#002" #045@ + IF DB-EXCEPTION IS 11 GO TO #04.@ + IF DB-EXCEPTION IS 12 GO TO #04.@ -%07. IF DB-STATUS = "0502100" GO TO #04.@ - IF DB-STATUS = "0503100" GO TO #04.@ PERFORM COUNTR. GO TO #06.@ * END %04. MOVE FALSE TO #20. MOVE #21 TO LFOUND.@ IF #21 = TRUE GO TO #05.@ * FAILURE RETURN MOVE #060 TO PPT. PERFORM FRTRN.@ GO TO #006.@ * SUCCESS RETURN %05. MOVE #070 TO PPT. PERFORM SRTTRN#037#200.@ GO TO #007.@ * ANY OCCURRENCE IS FOUND %06.@ - ACCEPT #30 FROM CURRENCY.@ + MOVE PGCS OF FCOM TO #30.@ * PRO-OR ANTI-NODE #042@ IF #30 = #010 GO TO #03. #042@ PERFORM GCOUNT. #095@ - GET #001. #095@ + GET CURRENT. #095@ IF #020 NEXT SENTENCE ELSE GO TO #04.@ IF #021 NEXT SENTENCE ELSE GO TO #04.@ (MOVE #061 #075 TO #062 IN R#067#200.@) IF #066 NEXT SENTENCE ELSE GO TO #04.@ MOVE #30 TO DBK. #005@ PERFORM RESULT. #005@ * LAST-NODE #043@ MOVE TRUE TO #21. GO TO #03. #043@ </pre>					

パターン名	NEXT	d #	s-d #	gf #	DG #	node #
	DML ブロック	5 0	1 0	*	*	*
	<pre> NEXT 00500010 ----- **** NEXT *** NEXT (Y037#200-%00) (%001 VIA %002)@ %01. MOVE %00 TO PT.@ + MOVE "FIND" TO WCON. + MOVE "%002" TO SET-NAME-W. PERFORM SET-SEARCH. + IF %20 = TRUE GO TO %02.@ + ADD 1 TO SET-MC (SET-P). + MOVE FALSE TO LFOUND. + MOVE TRUE TO %20. MOVE FALSE TO %21.@ - FIND FIRST %001 WITHIN %002.@ - GO TO %07.@ + GO TO %03.@ %02. IF LFOUND = TRUE MOVE TRUE TO %21.@ - FIND %001; DB-KEY IS %30. %040@ + MOVE %30 TO PGCS OF FCNM. %040@ + FIND CURRENT. %040@ %03.@ - FIND NEXT %001 WITHIN %002.@ + FIND NEXT "%001" WITHIN "%002".@ + IF DB-EXCEPTION IS 11 GO TO %04.@ + IF DB-EXCEPTION IS 12 GO TO %04.@ -%07. IF DB-STATUS = "0502100" GO TO %04.@ - IF DB-STATUS = "0503100" GO TO %04.@ + ADD 1 TO SET-OC (SET-P). + PERFORM COUNTR. GO TO %06.@ * END %04. MOVE FALSE TO %20. MOVE %21 TO LFOUND.@ + IF %21 = TRUE GO TO %05.@ * FAILURE! RETURN + MOVE %060 TO PPT. PERFORM FTRN.@ + GO TO %006.@ * SUCCESS RETURN %05. MOVE %070 TO PPT. PERFORM SRTN%037#200.@ + GO TO %007.@ * ANY OCCURRENCE IS FOUND %06.@ - ACCEPT %30 FROM CURRENCY.@ + MOVE PGCS OF FCNM TO %30.@ + PRD-OR ANIT-NODE %042@ + IF %30 = %010 GO TO %03. %042@ - GET %001. %095@ + GET CURRENT. %095@ + PERFORM GCOUNT. %095@ + IF %003 AND %004 NEXT SENTENCE ELSE GO TO %03.@ (+ MOVE %061 %075 TO %062 IN R%067#200.@) + IF %066 NEXT SENTENCE ELSE GO TO %03.@ + MOVE %30 TO DBK. PERFORM RESULT. %005@ * LAST NODE %043 + MOVE TRUE TO %21. GO TO %03. %043@ </pre>					

パターン名	CN	d #	s-d #	gf #	DQC #	node #
	DML ブロック	5 0	1 0	*	*	*
	<pre> CN 00500010 ----- ***** CN * CN (NOT LAST-NEXT-NODE) * THE N-TH CONFLUENT NODE(N >= 2)(%037**200-%00)(%001 VIA %002)@ %01. MOVE %00 TO PT.@ - IF %20 = TRUE GO TO %06.@ - MOVE TRUE TO %20.@ - FIND FIRST %001 WITHIN %002.@ - GO TO %07.@ -%06.@ - FIND NEXT %001 WITHIN %002.@ + FIND NEXT "%001" WITHIN "%002".@ + IF DB-EXCEPTION IS 11 GO TO %02.@ + IF DB-EXCEPTION IS 12 GO TO %02.@ -%07. IF DB-STATUS = "0502100" GO TO %02.@ - IF DB-STATUS = "0503100" GO TO %02.@ - PERFORM COUNTR. GO TO %03.@ * FAILURE RETURN (ENDSET) %02. MOVE FALSE TO LFOUND.@ - MOVE FALSE TO %20.@ - MOVE %060 TO PPT. PERFORM FRTRN.@ - GO TO %006.@ * ANY OCCURRENCL IS FOUND. -%03.@ - ACCEPT %30 FROM CURRENCY.@ + MOVE PGCS OF FCOM TO %30.@ * IF CURRENT DB-KEY = THE DB-KEY OF THE 1ST CN. - IF %30 = %010 GO TO %04. GO TO %01.@ %04. MOVE TRUE TO LFOUND.@ * LAST NODE %043@ - MOVE %070 TO PPT. PERFORM SRTN%037**200.%043@ - GO TO %007. %043@ </pre>					

パターン名	D - CALC	d #	s-d #	gf #	DQG #	node #
	DML ブロック	5 0	10	*	*	*
	<pre> D-CALC .00500010 ----- ***** D-CALC * D-CALC (#037**#200-#00) (#001)@ %02. MOVE #00 TO PT.@ PERFORM GNV#037**#200.#009@ IF MODEE = ENDD GU TO TERM#037**#200. MOVE CVAL#037**#200 TO #00R IN #001.@ - FIND ANY #001.@ + FIND ANY "#001".@ - IF DB-STATUS = "0502400" GO TO %02.@ + IF DB-EXCEPTION IS 13 GO TO %02.@ GO TO %03.@ %01.@ - FIND DUPLICATE #001.@ + FIND DUPLICATE "#001".@ - IF DB-STATUS = "0502400" GO TO %02.@ + IF DB-EXCEPTION IS 13 GO TO %02.@ %03.@ PERFORM COUNT. - GET #001. #096@ + GET CURRENT. #096@ PERFORM GCOUNT. #096@ IF #004 NEXT SENTENCE ELSE GO TO %01.@ SATISFIED. (MOVE #061 #075 TO #062 IN #R#067**#200.@) IF #066 NEXT SENTENCE ELSE GO TO %01.@ ACCEPT #30 FROM CURRENCY.@ + MOVE PGCS OF FCUM TO #30.@ MOVE #30 TO DBK. #005@ PERFORM RESULT. #005@ * LAST NODE #043@ MOVE #070 TO PPT. PERFORM SRTRN#037**#200. #042@ GO TO %01. #043@ </pre>					

パターン名	A - CALC	d #	s-d#	gf #	DQG#	node #
	DML ブロック	5 0	1 0	*	*	*
	<pre> A-CALC 00500010 ----- ***** * A-CALC *** ANY-CALC (#037##200-%00) (#001)@ %01. MOVE %00 TO PT.@ PERFORM GNV#037##200. #009@ IF MODEE = ENDD GO TO TERM#037##200.@ MOVE CVAL#037##200 TO #008 IN #001.@ FIND ANY #001.@ + FIND ANY "#001".@ + IF DB-EXCEPTION IS 13 GO TO #01.@ - IF DB-STATUS = "0502400" GO TO #01.@ PERFORM COUNTR. - GET #001. #096@ + GET CURRENT. #096@ PERFORM GCDUNT. #096@ IF #004 NEXT SENTENCE ELSE GO TO #01.@ * SATISFIED. (MOVE #061 #075 TO #062 IN R#067##200.@) IF #066 NEXT SENTENCE ELSE GO TO #01.@ - ACCEPT #30 FROM CURRENCY.@ + MOVE PGCS OF FCUM TO #30.@ MOVE #30 TO DBK. #005@ PERFORM RESULT. #005@ * LAST NODE #043@ MOVE #070 TO PPT. PERFORM SRTRN#037##200.#043@ GO TO #01. #043@ </pre>					

パターン名	DML - TERM1	d #	s-d#	gf #	DQG#	node #
	DML ブロック	5 0	2 0	*	*	—
	<pre> DMLTERM100500020 ----- * DML-TERM1 * TERMINATION OF DQG#037##200.@ TERM#037##200.@ CLOSE #035##200-F01. * IF C#037##200 > 3 THEN + PERFORM GETTIME. MOVE CTIME TO SRT1. + SORT SORT-#035##200-F01 ON ASCENDING SORT-ITEM-#035##200 + USING #035##200-F01 + GIVING #035##200-F01. + PERFORM GETTIME. MOVE CTIME TO SRT2. + COMPUTE SRTTM = SRT2 - SRT1. PERFORM CTIME31. MOVE GCNT TO SCNT. COMPUTE T13 = T13 - SRTTM. COMPUTE T31 = T13 + T31. COMPUTE T51 = T15 + T51. COMPUTE S01 = SCNT + S01. COMPUTE C1 = CNT + C1. COMPUTE G1 = GCNT + G1. COMPUTE G01 = GCNTD + G01. COMPUTE GSRTTM = GSRTTM + SRTTM. MOVE #037##200 TO I. MOVE C#037##200 TO CNTDQG. PERFORM PR-ACCO1. MOVE 0 TO T13 T15 SRTTM CNT GCNT GCNTD SCNT. IF CNTDQG = 0 #240@ MOVE 101 TO ENCODE IN WK-RELI0 #240@ DISPLAY "#035##200-F01 HAS NO TUPLE...STOP" #240@ GO TO REI0.#240@ </pre>					

パターン名	RREL	d #	s-d #	gf #	DQG #	node #
	DML ブロック	5 0	4 0	*	@	-
	<pre> RREL 00500030 *-----* * RREL * RESULT RELATION ¥039¥¥200 IS GENERATED@ * FROM INTERMEDIATE RESULTS. * * PERFORM GETTIME. MOVE CTIME TO T6. * MOVE 0 TO CTOTAL. MOVE 0 TO CF039¥¥200. * OPEN OUTPUT ¥039¥¥200-F01.@ </pre>					

パターン名	RREL 1	d #	s-d #	gf #	DQG #	node #
	DML ブロック	5 0	3 1	*	-	*
	<pre> RREL1 00500031 *-----* * RREL1 * IF C¥037¥¥200 = 0 GO TO R¥054¥¥200A.@ * MOVE 0 TO M¥035¥¥200. * OPEN INPUT ¥035¥¥200-F01.@ * R¥037¥¥200A. * READ ¥035¥¥200-F01 AT END@ * GO TO R¥037¥¥200B. * IF ¥035¥¥200-R01 = S¥035¥¥200 AND * M¥035¥¥200 = 1 GO TO R¥037¥¥200A. * MOVE 1 TO M¥035¥¥200. MOVE ¥035¥¥200-R01 TO S¥035¥¥200. * GO TO R¥037¥¥200C. *END * R¥037¥¥200B. * CLOSE ¥035¥¥200-F01. GO TO R¥054¥¥200A.@ * R¥037¥¥200C. * IF NOT (¥056) GO TO R¥037¥¥200A.¥059@ * * ¥057.@ * * MOVE ¥039¥¥200-R01 TO WK¥039¥¥200.¥058@ * DISPLAY ¥039¥¥200-R01. ¥233 * WRITE ¥039¥¥200-R01 IN ¥039¥¥200-F01 INVALID KEY ¥058@ * WRITE ¥039¥¥200-R01 IN ¥039¥¥200-F01.¥058@ * IF WST NOT = 10 ¥058@ * GO TO R¥037¥¥200D. ¥058@ * MOVE 451 TO ENCODE IN WK-RELID ¥058@ * DISPLAY "¥039¥¥200-F01 OVERFLOW" GO TO REND.¥058@ * R¥037¥¥200D. MOVE WK¥039¥¥200 TO ¥039¥¥200-R01.¥058@ * ADD 1 TO CTOTAL. ¥058@ * ADD 1 TO C¥039¥¥200.¥058 * GO TO R¥037¥¥200A.¥058@ </pre>					

パターン名	RREL11	d #	s-d #	gf #	DQG #	node #
	DML ブロック	5 0	3 1	*	—	*
	<pre> RREL11 00500031 *-----* * RREL1 * (* *057.e *) * DISPLAY #039**200-R01. + WHITE #039**200-R01 IN #039**200-F01 INVALID KEY - WRITE #039**200-R01 IN #039**200-F01. - IF WST NUT = 10 - GO TO R#037**200D. MOVE 451 TO ENDCODE IN WK-RELID #058e DISPLAY "#039**200-F01 OVERFLOW" GO TO REND. ADD 1 TO G1. ADD 1 TO CTOTAL. ADD -1 TO C#039**200. </pre>					

パターン名	RREL3	d #	s-d #	gf #	DQG #	node #
	DML ブロック	5 0	3 1	{0}	@	—
	<pre> RREL3 00500031 *-----* * RREL3 + MOVE 0 TO M#239**235. * OPEN INPUT #239**235-F#235. R#037**200A. READ #239**235-F#235 AT END@ GO TO R#037**200B. + IF #239**235-R#235 = S#239**235 AND + M#239**235 = 1 GO TO R#037**200A. + MOVE 1 TO M#239**235. MOVE #239**235-R#235 TO S#239**235. GO TO R#037**200C. *END R#037**200B. CLOSE #239**235-F#235. GO TO R#054**200A.e R#037**200C. IF NOT (#056) GO TO R#037**200A.#057e (*057.e) MOVE #039**200-R01 TO WK#039**200.#058e DISPLAY #039**200-R01.#234 ***** IF ONE NOT < #037**200 DISPLAY #039**200.#058e + WRITE #039**200-R01 IN #039**200-F01 INVALID KEY#058e - WRITE #039**200-R01 IN #039**200-F01.#058e - IF WST NUT = 10 #058e - GO TO R#037**200D. #058e MOVE 451 TO ENDCODE IN WK-RELID #058e DISPLAY "#039**200-F01 OVERFLOW" GO TO REND.#058e R#037**200D. MOVE WK#039**200 TO #039**200-R01.#058e ADD 1 TO CTOTAL. #058e ADD 1 TO C#039**200.#058e GO TO R#037**200A.#058e </pre>					

パターン名	RREL4	d #	s-d #	gf #	DQG #	node #
	DML ブロック	5 0	3 1	-	@	-
	<pre> RREL4 00500031 #----- # RREL4 + MOVE 0 TO M#239#235 N#239#235 E#239#235.@ OPEN INPUT #239#235-F#235. R#037#200A. IF E#239#235 = 1 GO TO R#037#200B. READ #239#235-F#235 AT END@ GO TO R#037#200B. + IF #239#235-R#235 = S#239#235 AND + M#239#235 = 1 GO TO R#037#200A. + MOVE 1 TO M#239#235. MOVE #239#235-R#235 TO S#239#235. GO TO R#037#200C. *END R#037#200B. MOVE 1 TO E#239#235.@ IF N#239#235 = 0 GO TO R#037#200E.@ CLOSE #239#235-F#235. GO TO R#054#200A.@ R#037#200E. MOVE 0 TO #242.@ MOVE 1 TO N#239#235.@ GO TO R#037#200F.@ R#037#200C. IF NOT (#256) GO TO R#037#200A.#059@ R#037#200F. MOVE 1 TO N#239#235.@ IF NOT (#257) GO TO R#037#200B.#258@ (#057.@) MOVE #039#200-R01 TO WK#039#200.#058@ DISPLAY #039#200-R01.#234 ***** IF ONE NOT < #037#200 DISPLAY #039#200.#058@ + WRITE #039#200-R01 IN #039#200-F01 INVALID KEY#058@ - WRITE #039#200-R01 IN #039#200-F01.#058@ - IF WST NOT = 10 #058@ - GO TO R#037#200D.#058@ MOVE 451 TO ENDCODE IN WK-RELD.#058@ DISPLAY "#039#200-F01 OVERFLOW" GO TO REND.#058@ R#037#200D. MOVE WK#039#200 TO #039#200-R01.#058@ ADD 1 TO CTOTAL.#058@ ADD 1 TO C#039#200.#058@ GO TO R#037#200A.#058@ </pre>					

パターン名	RREL20	d #	s-d#	gf #	DQG#	node #
	DML ブロック	5 0	3 2	*	@	—
	RREL20 00500032 *----- * RREL2 R000Y200A. CLOSE Y039Y200-F01.					

パターン名	RREL21	d #	s-d#	gf #	DQG#	node #
	DML ブロック	5 0	3 2	*	@	—
	RREL21 00500032 *----- * * IF CTOTAL > 1 * PERFORM GETTIME. MOVE CTIME TO SRT1. * SORT SORT-Y039Y200-F01 IN ASCENDING SORT-ITEM-Y039Y200 * USING Y039Y200-F01 * OUTPUT PROCEDURE NO-DUPY200. * PERFORM GETTIME. MOVE CTIME TO SRT2. * COMPUTE SRTTM = SRT2 - SRT1. * MOVE CTOTAL TO SCNT. * COMPUTE S01 = S01 + SCNT.					

パターン名	RREL3	d #	s-d#	gf #	DQG#	node #
	DML ブロック	5 0	3 2	*	@	—
	RREL22 00500032 *----- * RREL200. * PERFORM GETTIME. MOVE CTIME TO T7. * IF T2 = 0 THEN COMPUTE T2 = T7 - T6 * ELSE COMPUTE T2 = T7 - T2. * * COMPUTE T31 = T2 - T51 - (T7 - T6) - GSRTTM. * COMPUTE GSRTTM = GSRTTM + SRTTM. * COMPUTE T7 = T7 - T6 - SRTTM. * COMPUTE G1 = G1 + CTOTAL. * COMPUTE SCNT = SCNT + S01. * * DISPLAY "THE CARDINALITY = " CTOTAL. * DISPLAY "THE NUMB. OF LOCATED OCC = " C1. * DISPLAY "THE NUMB. OF ACCESSED OCC = " G01. * DISPLAY "THE NUMB. OF OUTPUT OCC = " G1. * DISPLAY "THE NUMB. OF SORT OCC = " S01. * DISPLAY "TOTAL TIME = " T2. * DISPLAY "THE TIME FOR LOCATING = " T31. * DISPLAY "THE TIME FOR OUTPUT = " T51. * DISPLAY "THE TIME FOR SORTING = " GSRTTM. * DISPLAY "THE TIME FOR MERGING = " T7. * COMPUTE TSORT = TSORT + GSRTTM. * COMPUTE TOUT = TOUT + T51. * COMPUTE TMERGE = TMERGE + T7. * COMPUTE TSCNT = TSCNT + S01. * COMPUTE TCNT = TCNT + C1. * COMPUTE TGCNT = TGCNT + G1. * COMPUTE TGCNT0 = TGCNT0 + G01. * MOVE 0 TO C1 G1 G01 S01 T2 T31 T51 T7 GSRTTM SRTTM.					

パターン名	STOP	d #	s-d #	gf #	DQG #	node #
	DML ブロック	7 0	9 0	—	—	—
	<pre> * STOP 00700090 *-----* * STOP * REUD. ***** TRANSACTION END.***** FINISH. PERFORM SET-CONNECT. COPY ENDPROC. STOP RUN. </pre>					

パターン名	AGR	d #	s-d #	gf #	DQG #	node #
	DML ブロック	5 0	4 0	*	@	—
	<pre> AGR 00500040 *-----* * AGGREGATE * PERFORM GETTIME. MOVE CTIME TO AGT1. * PERFORM AGRGT*200-RTN. * PERFORM GETTIME. MOVE CTIME TO AGT2. * COMPUTE AGRTMO = AGT2 - AGT1. * COMPUTE GAGRTM = GAGRTM + AGRTMO. * DISPLAY "THE TIME FOR AGGREGATE = " AGRTMO. * MOVE 0 TO AGRTMO. </pre>					

パターン名	GNV	d #	s-d #	gf #	DQG #	node #
	DML ブロック	8 0	0	*	*	—
	<pre> GNV 00800000 *-----* * GNV * GNV*037**200 - CVAL*037**200 = CALC VALUE@ * MODEE = ENDD * NOO * GNV*037**200 SECTION. IF CCV*037**200 NOT > 0 GO TO G1*037**200.* MOVE CV*037**200 (CCV*037**200) TO CVAL*037**200.@ SUBTRACT 1 FROM CCV*037**200.@ MOVE NOO TO MODEE. GO TO G2*037**200.* G1*037**200. MOVE ENDD TO MODEE. G2*037**200. EXIT. </pre>					

パターン名	S R T R N	d #	s-d #	g f #	D G G #	node #
	DML ブロック	8 0	1	*	*	-
	<pre> SRTN 00800001 ----- * ----- SRTN (SUCCESS RETURN) ----- * * PT = AT POINTER OF CURRENT NODE * PPT = AT POINTER OF SUCCESS RETURNED NODE * SRTN#037##200 SECTION. PERFORM LRM. SRTN1#037##200. IF PTND < PPTND GO TO SRTN3#037##200. IF RPPTND > LPPTND GO TO SRTN3#037##200. IF NLPTND = CSRT PERFORM OUT#037##200. * IF NLPTND NOT = CSRT AND NRPPTND NOT = 1 GO TO SRTN3#037##200. MOVE NRPPTND TO I. SRTN2#037##200. IF I > CSRT GO TO SRTN5#037##200. MOVE 0 TO CVPT IN SRSLT (I). IF I NOT = NRPPTND MOVE 0 TO LENGTT IN SRSLT (I). ADD 1 TO I. GO TO SRTN2#037##200. SRTN3#037##200. ADD NRPPTND, 1 GIVING I. SRTN4#037##200. IF I NOT > NLPTND MOVE 0 TO LENGTT IN SRSLT (I) ADD 1 TO I GO TO SRTN4#037##200. SRTN5#037##200. EXIT. </pre>					

パターン名	SETPFM	d #	s-d #	gf #	DQG #	node #
	DML ブロック	8 0	2	—	—	—
<pre> SETPFM 00800002 #----- # SET-SEARCH # # INPUT : SET-NAME-W & WCOM (FIND/APPD/DELT) # OUTPUT : SET-P & MAX-SET # +SET-SEARCH SECTION. + MOVE 1 TO I. +SET-TOP. + IF I > MAX-SET + MOVE SET-NAME-W TO SET-NAME (I) + MOVE WCOM TO SET-COMND (I) + MOVE I TO SET-P MAX-SET + ELSE + IF SET-NAME (I) = SET-NAME-W AND + SET-COMND (I) = WCOM + MOVE I TO SET-P + ELSE + ADD 1 TO I + GO TO SET-TOP. +SET-SRCH-EXIT. + EXIT. #----- +SET-CONNECT SECTION. + OPEN OUTPUT PERFM-F. + MOVE 1 TO I. +SET-LOOP. + IF I > MAX-SET GO TO SET-CLS. + MOVE SETC (I) TO PERFM-R. + DISPLAY PERFM-R. + WRITE PERFM-R INVALID KEY + DISPLAY "PERFM-F OVERFLOW" GO TO SET-CLS. + ADD 1 TO I. GO TO SET-LOOP. +SET-CLS. + CLOSE PERFM-F. +SET-CONNECT-EXIT. + EXIT. </pre>						

パターン名	SUB	d #	s-d#	gf #	DQG#	node#
	DML ブロック	8 0	2	—	—	—
	SUB 00800002 COPY MMUISUB. ERR-SECTION SECTION. ERR-RTN. + DISPLAY "DB-EXCETION ERROR". - DISPLAY "DB-STATUS = " DB-STATUS. ERR-EXT. EXIT.					

パターン名	OUTP 1	d #	s-d#	gf #	DQG#	node#
	DML ブロック	8 2	0	*	*	—
	OUTP1 00820000 ----- * OUTP1 * OUT*037**200@ * OUTPUT TO *035**200-F FROM SRSLT.@ * OUT*037**200 SECTION.@ PERFORM GETTIME. MOVE CTIME TO T4. MOVE 0 TO OI OJ. OI1*037**200.@ ADD 1 TO OI. IF OI > CVPT IN SRSLT (CSRT) GO TO OI1*037**200. OI2*037**200.@					

パターン名	OUTP 2	d #	s-d#	gf #	DQG#	node#
	DML ブロック	8 2	1	*	*	—
	OUTP2 00820001 ----- * OUTP2 * ADD 1 TO OJ. (*037**200-%00)@ IF OI NOT > 1 GO TO O*037**200%00A.@ COMPUTE I = OI - 1. IF DBKEY IN STDBK (OJ, OI) = DBKEY IN STDBK (OJ, I) GO TO O*037**200%00B.@ O*037**200%00A. - FIND *001:DB-KEY IS DBKEY IN STDBK (OI, OJ).@ + MOVE DBKEY IN STDBK (OJ, OI) TO PGCS OF PCOM. + GET *001.@ + GET CURRENT. PERFORM GCOUNTP. MOVE DBKEY IN STDBK (OJ, OI) TO *016%00 IN *035**200-R01.? (MOVE *013 IN **001 TO *014%00 IN *035**200-R01.??) O*037**200%00B.					

パターン名	OUTP3	d #	s-d #	gf #	DQG #	node #
	DML ブロック	8 2	2	*	*	—
	<pre> OUTP3 00820002 #----- # OUTP3 # GET AND WRITE TO %035%200-F01 (%037%200-%00).% # PROCESSING OF NON-SUPPORTED JOINS. # IF %036 NEXT SENTENCE ELSE GO TO UTM%037%200.%5916 # MOVE %035%200-R01 TO WK%035%200. + WRITE %035%200-R01 INVALID KEY% - WRITE %035%200-R01.% - IF WST%037%200 NOT = 10 GO TO UTM%037%200.% # MOVE 451 TO ENDCODE IN WK-RELID # DISPLAY "%035%200-F01 OVERFLOW"% # GO TO UTL%037%200.% # ADD 1 TO C%037%200. # MOVE WK%035%200 TO %035%200-R01. # UTM%037%200.% # MOVE 0 TO UJ. GO TO UT1%037%200.% # UTL%037%200.% # PERFORM CTJME51. # DTEX%037%200.% # EXIT. </pre>					

パターン名	NO - DUP	d #	s-d #	gf #	DQG #	node #
	DML ブロック	8 3	0	*	—	—
	<pre> NO-DUP 00830000 #----- +NO-DUP%200 SECTION. +NO-DUP%200. + OPEN OUTPUT %039%200-F01. + MOVE 0 TO MODES. + MOVE SPACE TO S%039%200. +NO-DUP1%200. + RETURN SORT=%039%200-F01 AT END GO TO NO-DUP2%200. + MOVE SORT=%039%200-R01 TO %039%200-R01. + IF %039%200-R01 = S%039%200 AND + MODES = 1 GO TO NO-DUP1%200. + MOVE 1 TO MODES. MOVE %039%200-R01 TO S%039%200. + DISPLAY %039%200-R01. + WRITE %039%200-R01 FROM SORT-%039%200-R01. + GO TO NO-DUP1%200. +NO-DUP2%200. CLOSE %039%200-F01. +NO-DUP3%200. EXIT. </pre>					

パターン名	UPDT	d #	s-d #	gf #	DQG #	node #
	DML ブロック	7 0	0	—	—	—
	<pre> UPDT 00700000 #..... # UPDT # # UPDATE EXECUTION # # PERFORM UPDT. </pre>					

パターン名	UP - TIME	d #	s-d#	gf#	DQG#	node#
	DML ブロック	8 0	4	—	—	—
	UP-TIME 00800004 * * UP-TIME * * UPTIME TDO : START TIME * UP-TIME. PERFORM GETTIME. MOVE CTIME TO T01. COMPUTE T01 = T01 - T00. UP-TIME0. EXIT.					

パターン名	UPDTM1	d #	s-d#	gf#	DQG#	node#
	DML ブロック	9 0	0	—	—	—
	UPDTM1 00900000 * * UPDTM1 * UPDATE RELATION. * UPDT SECTION. PERFORM GETTIME. MOVE CTIME TO TDO. OPEN INPUT #039#*201-F01. UPDT1. READ #039#*201-F01 AT END GO TO UPDT-L. * ONE TUPLE IS READ INTO.					

パターン名	UPDTM3	d #	s-d#	gf#	DQG#	node#
	DML ブロック	9 0	9	—	—	—
	UPDTM3 00900009 * * UPDTM3 * END * COPY DBEXCEPT. UPDT-L. CLOSE #039#*201-F01. PERFORM UP-TIME. DISPLAY "TIME FOR UPDATE = " T01. UPDT-L0. EXIT.					

パターン名	DEL - E	d #	s-d#	gf#	DQG#	node#
	DML ブロック	9 0	0	—	—	—
	DEL-E 00900000 * * DEL-E * DELETION OF E-RELATION TUPLE. - FIND #900; DB-KEY IS #901 IN #039#201-R01. + MOVE #901 IN #039#201-R01 TO PGCS OF FCOM.# * FIND CURRENT.					

パターン名	DEL - E4	d #	s-d #	gf #	DQG #	node #
	DML ブロック	9 0	2	—	—	—
	DEL-E4 00900002 *..... * DEL-E4 * DELETION OF E-RELATION TUPLE. * ERASE *900 PERMANENT.@ * ERASE "¥900".@ + IF DB-EXCEPTION IS 13 GO TO DBER13. + MOVE "DELT" TO WCDM. MOVE "¥900" TO SET-NAME-W. + PERFORM SET-SEARCH. ADD 1 TO SET-MC (SET-P). - ERASE *900 ALL.@ - IF DB-STATUS = "0407200" PERFORM ERR-SECTION. GO TO UPDT1.					

パターン名	DEL - MA	d #	s-d #	gf #	DQG #	node #
	DML ブロック	9 0	1	—	—	—
	DEL-MA 00900001 *..... * DEL-MA * DELETION OF R-RELATION TUPLE * ¥900 IS M/A SET-TYPE @ - FIND ¥911; DB-KEY IS ¥901 IN ¥039¥201-R01. + MOVE ¥901 IN ¥039¥201-R01 TO PGCS (IF FCUM.@ + FIND CURRENT. * ERASE ¥911 PERMANENT. @ + ERASE "¥911".@ + IF DB-EXCEPTION IS 13 GO TO DBER13. - ERASE ¥911 ALL. @ - IF DB-STATUS = "0407200" PERFORM ERR-SECTION. GO TO UPDT1.					

パターン名	DEL - MA1	d #	s-d #	gf #	DQG #	node #
	DML ブロック	9 0	0	—	—	—
	DEL-MA1 00900000 * DELETION OF R-RELATION TUPLE - FIND ¥911; DB-KEY IS ¥901 IN ¥039¥201-R01. + MOVE ¥901 IN ¥039¥201-R01 TO PGCS OF FCUM.@ + FIND CURRENT.					

パターン名	DEL - MA2	d #	s-d #	gf #	DQG #	node #
	DML ブロック	9 0	1	—	—	—
	DEL-MA2 00900001 EL01¥070. + FIND NEXT "¥001" WITHIN "¥002". + IF DB-EXCEPTION IS 11 GO TO EL02¥070. + IF DB-EXCEPTION IS 12 GO TO EL02¥070.					

パターン名	DEL - MA 3	d #	s-d #	gf #	DQG #	node #
	DML ブロック	9 0	2	-	-	-
	DEL-MA3 00900001 + ERASE "¥001". + MOVE "DELT" TO WCOM. MOVE "¥001" TO SET-NAME-W. + PERFORM SET-SEARCH. ADD 1 TO SET-MC (SET-P). GO TO ELO1¥070. ELO2¥070.					

パターン名	DEL - MA 4	d #	s-d #	gf #	DQG #	node #
	DML ブロック	9 0	2	-	-	-
	DEL-MA4 00900002 + ERASE "¥911". + MOVE "DELT" TO WCOM. MOVE "¥911" TO SET-NAME-W. + PERFORM SET-SEARCH. ADD 1 TO SET-MC (SET-P). GO TO UPDT1.					

パターン名	DEL - OM	d #	s-d #	gf #	DQG #	node #
	DML ブロック	9 0	1	-	-	-
	DEL-OM 00900001 *..... * DEL-OM * DELETION OF R-RELATION TUPLE * ¥900 IS O/M SET-TYPE. @ - FIND ¥911; DB-KEY IS ¥901 IN ¥039¥201-R01. + MOVE ¥901 IN ¥039¥201-R01 TO PGCS OF FCOM. @ + FIND CURRENT. + DISCONNECT "¥911" FROM "¥900". @ + IF DB-EXCEPTION IS 02 GO TO DBER02. + IF DB-EXCEPTION IS 04 NEXT SENTENCE. - DISCONNECT ¥911 FROM ¥900. @ - IF DB-STATUS = "0304800" PERFORM ERR-SECTION. - IF DB-STATUS = "0308200" PERFORM ERR-SECTION.					

パターン名	APP - S	d #	s-d #	gf #	DQG #	node #
	DML ブロック	9 0	1	-	-	-
	<pre> APP-S 00900001 * * APP-S * APPEND TO R-RELATION * #900 IS D/M SET-TYPE @ - FIND #910; DB-KEY IS #901 IN #039#201-R01. - FIND #911; DB-KEY IS #901 IN #039#200-R01. + MOVE #901 IN #039#201-R01 TO DBM.@ + MOVE #901 IN #039#201-R01 TO DBD.@ + MOVE DBU TO PGCS OF FCDM. + FIND CURRENT. + MOVE DBM TO PGCS OF FCDM. + CONNECT "#911" TO "#900".# + IF DB-EXCEPTION IS 14 GO TO DNER16. + IF DB-EXCEPTION IS 16 GO TO DNER16. - CONNECT #911 TO #900.# - IF DB-STATUS = "0205100" PERFORM ERR-SECTION. - IF DB-STATUS = "0208100" PERFORM ERR-SECTION. GO TO UPDT1. </pre>					

パターン名	APP - E	d #	s-d#	gf#	DQG#	node#
	DML ブロック	90	1	-	-	-
	<pre> APP-E 00900001 #..... * APP-E * APPEND TO E-RELATION * LOCATE OWNERS OF #900.# (- FIND #915; DB-KEY IS #901 IN ##039##201-R01.) * CREATE NEW TUPLE (MOVE #902 IN ##039##201-R01 TO #903 IN ##900.#) + STORE "#900".# + IF DB-EXCEPTION IS 14 GO TO DBER14. + IF DB-EXCEPTION IS 18 GO TO APP-E01. + IF DB-EXCEPTION IS 11 GO TO DBER11. + DISPLAY "DB-EXCEPTION NOT = 14, 18, 11". + MOVE "APPD" TO WCOM. MOVE "#900" TO SET-NAME-W. + PERFORM SET-SEARCH. ADD 1 TO SET-MC (SET-P). + GO TO UPDT1. +APP-E01. + MOVE "APPD" TO WCOM. MOVE "#900" TO SET-NAME-W. + PERFORM SET-SEARCH. ADD 1 TO SET-MC (SET-P). + STORE #900.# + IF DB-STATUS = "1502300" PERFORM ERR-SECTION. + IF DB-STATUS = "1505100" PERFORM ERR-SECTION. + IF DB-STATUS = "1580200" PERFORM ERR-SECTION. + MOVE PGCS OF FCOM TO UDBK. (MOVE #901 IN ##039##201-R01 TO PGCS OF FCOM. FIND CURRENT.#904 MOVE UDBK TO PGCS OF FCOM.#904 CONNECT "##900" TO "#918".#904 CONNECT ##900 TO #918. #904 + IF DB-EXCEPTION IS 14 GO TO APP-ERR.#904 + IF DB-EXCEPTION IS 16 GO TO APP-ERR.#904) GO TO UPDT1. +APP-ERR. + MOVE UDBK TO PGCS OF FCOM. + FIND CURRENT. + ERASE "#900".# + DISPLAY "DB-EXCEPTION HAPPENED IN APP-E.". + GO TO UPDT1. </pre>					

パターン名	APP - L	d #	s-d #	gf #	DQG #	node #
	DML ブロック	9 0	1	-	-	-
	<pre> APP-L 00900001 #..... # APP-L # APPEND TO R-RELATION # *900 IS A LINK RECORD-TYPE * # LOCATE THE OWNERS. + MOVE *931 IN *039*201-R01 TO PGCS OF FCOM. + FIND CURRENT. # ----- FIND LINK-RECORD-TYPE ----- + FIND FIRST "900" WITHIN "936". + IF DB-EXCEPTION IS 11 GO TO APP-L04. + IF DB-EXCEPTION IS 12 GO TO APP-L04. + GO TO APP-L03. +APP-L02. + FIND NEXT "900" WITHIN "937". + IF DB-EXCEPTION IS 11 GO TO APP-L04. + IF DB-EXCEPTION IS 12 GO TO APP-L04. +APP-L03. + MOVE PGCS OF FCOM TO UDBK. # ----- SEARCH ALL OWNER RECORD-TYPE ---- (+ FIND OWNER WITHIN "936". + IF DB-EXCEPTION IS 12 GO TO APP-L02. *934 + IF PGCS OF FCOM NOT = *931 IN *039*201-R01 *934 + GO TO APP-L02. *934) # + MOVE UDBK TO PGCS OF FCOM. + GET CURRENT. (+ IF *932 IN *039*201-R01 NOT = @ + *933 IN *900 GO TO APP-L02.) + DISPLAY "LINK-RECORD DUPLICATE ERROR". + MOVE 012 TO ENDCODE IN WK-RELID. + ADD 1 TO ERRCNT IN WK-RELID. + GO TO UPDT1. +APP-L04. (+ FIND *915: DB-KEY IS *901 IN *039*201-R01.) # CREATE A NEW TUPLE (+ MOVE *902 IN *039*201-R01 TO *903 IN *900.@) + STORE "900".@ + STORE *900.@ + IF DB-EXCEPTION IS 18 GO TO APP-L05. + IF DB-EXCEPTION IS 11 GO TO DBER11. + IF DB-EXCEPTION IS 14 GO TO DBER14. + IF DB-STATUS = "1502300" PERFORM ERR-SECTION. + IF DB-STATUS = "1505100" PERFORM ERR-SECTION. + IF DB-STATUS = "1580200" PERFORM ERR-SECTION. + GO TO UPDT1. +APP-L05. + MOVE PGCS OF FCOM TO UDBK. + MOVE "APPD" TO WCOM. MOVE "900" TO SET-NAME-W. + PERFORM SET-SEARCH. ADD 1 TO SET-MC (SET-P). (+ MOVE *901 IN *039*201-R01 TO PGCS OF FCOM.@ + FIND CURRENT. *904@ + MOVE UDBK TO PGCS OF FCOM.*904 + CONNECT "900" TO "916". + CONNECT *900 TO *916.) + GO TO UPDT1. </pre>					

パターン名	REP - E	d #	s-d #	gf #	DQG #	node #
	DML ブロック	9 0	1	-	-	-
	<pre> REP-E 00900001 * * REP-E * REPLACE E-RELATION * LOCATE THE TUPLE - FIND #900: DB-KEY IS #901 IN #039#201-R01. - GET #900.@ + MOVE #901 IN #039#201-R01 TO PGCS OF FCUM.# + GET CURRENT. * NEW TUPLES ARE CREATED (MOVE #902 IN ##039##201-R01 ,TO #903 IN ##900.*) * REPLACE + MODIFY "#900".@ - MODIFY #900 INCLUDING ALL MEMBERSHIP.# - IF DB-STATUS = "1105100" PERFORM ERR-SECTION. - IF DB-STATUS = "1102300" PERFORM ERR-SECTION. GO TO UPDT1. </pre>					

パターン名	REP - S	d #	s-d #	gf #	DQG #	node #
	DML ブロック	9 0	1	-	-	-
	<pre> REP-S 00900001 * * REP-S * REPLACE R-RELATION * #900 IS A SET-TYPE - FIND #910: DB-KEY IS #901 IN #039#201-R01. - FIND #911: DB-KEY IS #901 IN #039#201-R01. + MOVE #901 IN #039#201-R01 TO DBM. + MOVE #901 IN #039#201-R01 TO DBU. + MOVE DBM TO PGCS OF FCUM. + FIND CURRENT. + DISCONNECT "#911" FROM "#900". + MOVE DBU TO PGCS OF FCUM. + FIND CURRENT. + MOVE DBM TO PGCS OF FCUM. + CONNECT "#911" TO "#900". + IF DB-EXCEPTION IS 14 GO TO DBER16. + IF DB-EXCEPTION IS 16 GO TO DBER16. - MODIFY #911 ONLY #900 - MEMBERSHIP. - IF DB-STATUS = "1102300" PERFORM ERR-SECTION. - IF DB-STATUS = "1108300" PERFORM ERR-SECTION. GO TO UPDT1. </pre>					

パターン名	REP - L	d #	s-d #	gf #	DQG #	node #
	DML ブロック	9 0	1	—	—	—
	<pre> *..... # REP-L # REPLACE R-RELATION TUPLE # ¥900 IS A LINK RECORD-TYPE @ # LOCATE OWNERS # (# - FIND ¥915; DB-KEY IS ¥901 IN ¥¥039¥¥201-R01. #) # + MOVE ¥901 IN ¥¥039¥¥201-R01 TO PGCS OF FCUM. # + GET CURRENT. # (# + MOVE ¥902 IN ¥¥039¥¥201-R01 TO ¥903 IN ¥¥900. #) # + MOVE PGCS OF FCUM TO UDBK. # + MODIFY "¥900". # - GET ¥900. # { # + DISCONNECT "¥¥900" FROM "¥916". #) # (# + MOVE ¥901 IN ¥¥039¥¥201-R01 TO PGCS OF FCUM. # + FIND CURRENT. ¥904@ # + MOVE UDBK TO PGCS OF FCUM.¥904 # + CONNECT "¥¥900" TO "¥917". #) # - MODIFY ¥900 INCLUDING ¥916 # - ,¥916 # - ,¥916 # - ,¥916 # - ,¥916 # - ,¥916 # - ,¥916 # - MEMBERSHIP. # - GO TO UPDT1. </pre>					

パターン名	COUNT - S	d #	s-d #	gf #	DQG #	node #
	DML ブロック	8 1	0	*	—	—
	<pre> COUNT-S 00#10000 #-----* # COUNT-SIMPLE (¥200) ----* # AGRGT¥200-RTN SECTION. # + MOVE C¥039¥200 TO P-V-¥200. # + DISPLAY "COUNT-S P-V-¥200 = " P-V-¥200. # AGRGT¥200-EXIT. # EXIT. </pre>					

パターン名	AVER	d #	s-d #	gf #	DQG #	node #
	DML ブロック	8 1	0	*	—	—
	<pre> AVER 00810000 *-----* * ----- AVER (#200) ----- * AGRG#200-RTN SECTION. IF C#039#200 NOT > 0 GO TO AG#200-EXIT. OPEN INPUT #039#200-F01. MOVE 0 TO ENDF. OPEN OUTPUT #039#200-F#200. READ #039#200-F01 AT END GO TO AG#200-4. MOVE 0 TO P-V-#200. MOVE 0 TO I. AG#200-1. (MOVE #207 IN #039#200-R01 TO #208 IN AG-R-#200.) ADD #205 IN #039#200-R01 TO P-V-#200. ADD 1 TO I. AG#200-2. READ #039#200-F01 AT END MOVE 9 TO ENDF GO TO AG#200-3. (IF #209 IN #039#200-R01 NOT = 0 #210 IN AG-R-#200 GO TO AG#200-3.) GO TO AG#200-1. AG#200-3. (MOVE #211 IN AG-R-#200 TO #212 IN #039#200-R#200.) COMPUTE #205 IN #039#200-R#200 = P-V-#200 / I. DISPLAY "AVER....." #039#200-R#200. WRITE #039#200-R#200 INVALID KEY MOVE 451 TO ENDCODE IN WK-RELID. GO TO AG#200-4. MOVE 0 TO P-V-#200. MOVE 0 TO I. IF ENDF = 0 GO TO AG#200-1. AG#200-4. CLOSE #039#200-F01 #039#200-F#200. AG#200-EXIT. EXIT. </pre>					

パターン名	AVER - S	d #	s-d #	gf #	DQG #	node #
	DML ブロック	8 1	0	*	—	—
	<pre> AVER-S 00810000 *-----* * ----- AVER-SIMPLE (#200) ----- * AGRG#200-RTN SECTION. MOVE 0 TO P-V-#200. IF C#039#200 NOT > 0 GO TO AG#200-EXIT. OPEN INPUT #039#200-F01. AG#200-1. READ #039#200-F01 AT END GO TO AG#200-2. ADD #205 IN #039#200-R01 TO P-V-#200. GO TO AG#200-1. AG#200-2. COMPUTE P-V-#200 = P-V-#200 / C#039#200. CLOSE #039#200-F01. DISPLAY "AVEG-S P-V-#200 = " P-V-#200. AG#200-EXIT. EXIT. </pre>					

パターン名	SUM	d #	s-d #	gf #	DQG #	node #
	DML ブロック	8 1	0	*	—	—
	<pre> SUM 00810000 *-----* * SUM (#200) -----* * AGRGT#200-RTN SECTION. * IF C#039#200 NOT > 0 GO TO AG#200-EXIT. * OPEN INPUT #039#200-F01. MOVE 0 TO ENDF. * OPEN OUTPUT #039#200-F#200. * READ #039#200-F01 AT END GO TO AG#200-4. * MOVE 0 TO P-V-#200. * * AG#200-1. * (* MOVE #207 IN #039#200-R01 TO #208 IN AG-R-#200. *) * ADD #205 IN #039#200-R01 TO P-V-#200. * AG#200-2. * READ #039#200-F01 AT END MOVE 9 TO ENDF GO TO AG#200-3. * (* IF #209 IN #039#200-R01 NOT = @ * #210 IN AG-R-#200 GO TO AG#200-3. *) * GO TO AG#200-1. * AG#200-3. * (* MOVE #211 IN AG-R-#200 TO #212 IN #039#200-R#200. *) * MOVE P-V-#200 TO #205 IN #039#200-R#200. * DISPLAY "SUM....." #039#200-R#200. * WRITE #039#200-R#200 INVALID KEY * MOVE 451 TO ENDCODE IN WK-RELID * GO TO AG#200-4. * MOVE 0 TO P-V-#200. * IF ENDF = 0 GO TO AG#200-1. * AG#200-4. * CLOSE #039#200-F01 #039#200-F#200. * AG#200-EXIT. * EXIT. </pre>					

パターン名	SUM - S	d #	s-d #	gf #	DQG #	node #
	DML ブロック	8 1	0	*	—	—
	<pre> SUM-S 00810000 *-----* * SUM-SIMPLE (#200) -----* * AGRGT#200-RTN SECTION. * MOVE 0 TO P-V-#200. * IF C#039#200 NOT > 0 GO TO AG#200-EXIT. * OPEN INPUT #039#200-F01. * * AG#200-1. * READ #039#200-F01 AT END GO TO AG#200-2. * ADD #205 IN #039#200-R01 TO P-V-#200. * GO TO AG#200-1. * AG#200-2. * DISPLAY "SUM-S. P-V-#200 = " P-V-#200. * CLOSE #039#200-F01. * AG#200-EXIT. * EXIT. </pre>					

パターン名	MAX	d #	s-d#	gf #	DQG#	node #
	DML ブロック	8 1	0	*	—	—
	<pre> MAX 00810000 *-----* * MAX (#200) -----* AGRG1#200-RTN SECTION. IF C#039#200 NOT > 0 GO TO AG#200-EXIT. OPEN INPUT #039#200-F01. MOVE 0 TO ENDF. OPEN OUTPUT #039#200-F#200. READ #039#200-F01 AT END GO TO AG#200-4. AG#200-1. (MOVE #207 IN #039#200-R01 TO #208 IN AG-R-#200.) MOVE #205 IN #039#200-R01 TO P-V-#200. AG#200-2. READ #039#200-F01 AT END MOVE 9 TO ENDF GO TO AG#200-3. (IF #209 IN #039#200-R01 NOT = @ #210 IN AG-R-#200 GO TO AG#200-3.) IF #205 IN #039#200-R01 > P-V-#200 MOVE #205 IN #039#200-R01 TO P-V-#200. GO TO AG#200-2. AG#200-3. (MOVE #211 IN AG-R-#200 TO #212 IN #039#200-R#200.) MOVE P-V-#200 TO #205 IN #039#200-R#200. DISPLAY "MAX....." #039#200-R#200. WRITE #039#200-R#200 INVALID KEY MOVE 451 TO ENDCODE IN WK-RELID GO TO AG#200-4. IF ENDF = 0 GO TO AG#200-1. AG#200-4. CLOSE #039#200-F01 #039#200-F#200. AG#200-EXIT. EXIT. </pre>					

パターン名	MAX - S	d #	s-d#	gf #	DQG#	node #
	DML ブロック	8 1	0	*	—	—
	<pre> MAX-S 00810000 *-----* * MAX-SIMPLE (#200) -----* AGRG1#200-RTN SECTION. MOVE 999999999 TO P-V-#200. IF C#039#200 NOT > 0 @ GO TO AG#200-EXIT. OPEN INPUT #039#200-F01. AG#200-1. READ #039#200-F01 AT END GO TO AG#200-2. MOVE #205 IN #039#200-R01 TO P-V-#200. GO TO AG#200-1. AG#200-2. CLOSE #039#200-F01. DISPLAY "MAX-S P-V-#200 = " P-V-#200. AG#200-EXIT. EXIT. </pre>					

パターン名	MIN	d #	s-d #	gf #	DQG #	node #
	DML ブロック	8 1	0	*	—	—
	<pre> MIN 00810000 #-----# #----- MIN (*200) -----# AGRGY200-RTN SECTION. IF C*039*200 NOT > 0 GO TO AG*200-EXIT. OPEN INPUT *039*200-F01. MOVE 0 TO ENDF. OPEN OUTPUT *039*200-F*200. READ *039*200-F01 AT END MOVE 9 TO ENDF GO TO AG*200-4. AG*200-1. (MOVE *207 IN *039*200-R01 TO *208 IN AG-R-*200.) MOVE *205 IN *039*200-R01 TO P-V-*200. AG*200-2. READ *039*200-F01 AT END MOVE 9 TO ENDF GO TO AG*200-3. (IF *209 IN *039*200-R01 NOT = @ *210 IN AG-R-*200 GO TO AGY200-3.) IF *205 IN *039*200-R01 < P-V-*200 MOVE *205 IN *039*200-R01 TO P-V-*200. GO TO AG*200-2. AG*200-3. (MOVE *211 IN AG-R-*200 TO *212 IN *039*200-R*200.) MOVE P-V-*200 TO *205 IN *039*200-R*200. DISPLAY "MIN....." *039*200-R*200. WRITE *039*200-R*200 INVALID KEY MOVE 451 TO ENDCODE IN WK-RELID GO TO AG*200-4. IF ENDF = 0 GO TO AG*200-1. AG*200-4. CLOSE *039*200-F01 *039*200-F*200. AG*200-EXIT. EXIT. </pre>					

パターン名	MIN - S	d #	s-d #	gf #	DQG #	node #
	DML ブロック	8 1	0	*	—	—
	<pre> MIN-S 00810000 #-----# #----- MIN-SIMPLE (*200) -----# AGRGY200-RTN SECTION. MOVE -999999999 TO P-V-*200. IF C*039*200 NOT > 0 @ GO TO AGRGT*200-EXIT. OPEN INPUT *039*200-F01. READ *039*200-F01 AT END GO TO AGRGT*200-EXIT. MOVE *205 IN *039*200-R01 TO P-V-*200. CLOSE *039*200-F01. DISPLAY "MIN-S P-V-*200 = " P-V-*200. AGRGY200-EXIT. EXIT </pre>					

パターン名	COUNT	d #	s-d #	gf #	DQC #	node #
	DML ブロック	8 1	0	*	—	—
	COUNT 00810000 *-----* * COUNT (Y200) -----* AGRGTY200-RTN SECTION. IF C*039Y200 NOT > 0 GO TO AGY200-EXIT. OPEN INPUT Y039Y200-F01. OPEN OUTPUT Y039Y200-FY200. MOVE 0 TO P-V-Y200. READ Y039Y200-F01 AT END GO TO AGY200-3. AGY200-1. (MOVE Y207 IN Y*039Y200-R01 TO Y208 IN AG-H-Y200.) ADD 1 TO P-V-Y200. READ Y039Y200-F01 AT END MOVE 1 TO I GO TO AGY200-2. MOVE 0 TO I. (IF Y209 IN Y*039Y200-R01 NOT = @ Y210 IN AG-R-Y200 GO TO AGY200-2.) GO TO AGY200-1. AGY200-2. (MOVE Y211 IN AG-R-Y200 TO Y212 IN Y039Y200-RY200.) MOVE P-V-Y200 TO Y205 IN Y039Y200-RY200. DISPLAY "COUNT....." Y039Y200-RY200. WRITE Y039Y200-RY200 INVALID KEY MOVE 451 TO ENDCODE IN WK-RELID GO TO AGY200-3. MOVE 0 TO P-V-Y200. IF I = 0 GO TO AGY200-1. AGY200-3. CLOSE Y039Y200-F01 Y039Y200-FY200. AGY200-EXIT. EXIT.					

パターン名	ANY - S	d #	s-d #	gf #	DQC #	node #
	DML ブロック	8 1	0	*	—	—
	ANY-S 00810000 *-----* * ANY-SIMPLE (Y200) -----* AGRGTY200-RTN SECTION. IF C*039Y200 > 1 MOVE 1 TO P-V-Y200 ELSE MOVE 0 TO P-V-Y200. DISPLAY "ANY-S P-V-Y200 = " P-V-Y200. AGRGTY200-EXIT. EXIT.					

パターン名	ANY	d #	s-d #	gf #	DQG #	node #
	DML ブロック	8 1	0	x	—	—
	<pre> ANY 00810000 -----* # ----- ANY (#200) ----- AGRGT#200-RTN SECTION. IF C#039#200 NDT > 0 GO TO AG#200-EXIT. OPEN INPUT #039#200-F01. MOVE 0 TO ENDF. OPEN OUTPUT #039#200-F#200. READ #039#200-F01 AT END GO TO AG#200-4. AG#200-1. (MOVE #207 IN #039#200-R01 TO #208 IN AG-R-#200.) MOVE 1 TO P-V-#200. AG#200-2. READ #039#200-F01 AT END MOVE 9 TO ENDF GO TO AG#200-3. (IF #209 IN #039#200-R01 NDT = @ #210 IN AG-R-#200 GO TO AG#200-3.) GO TO AG#200-2. AG#200-3. (MOVE #211 IN AG-R-#200 TO #212 IN #039#200-R#200.) MOVE P-V-#200 TO #205 IN #039#200-R#200. DISPLAY "ANY....." #039#200-R#200. WRITE #039#200-R#200 INVALID KEY MOVE 451 TO ENDCODE IN WK-RELI0 GO TO AG#200-4. IF ENDF = 0 GO TO AG#200-1. AG#200-4. CLOSE #039#200-F01 #039#200-F#200. AG#200-EXIT. EXIT. </pre>					

COPY LIBRARY

パターン名	RESLTID	d #	s-d #	gf #	DQG #	node #
	DML ブロック					
	<pre> *----- * ---- RESULT RELATION ID FILE FD RELID-F LABEL RECORD IS STANDARD. 01 RELID-R. 03 LCSNAME PIC X(16). 03 USERID PIC X(11). 03 RELID PIC X(23). 03 ENDCODE PIC 9(04). 03 STCODE PIC 9(08). 03 ERRCNT PIC 9(08). 03 FINDCNT PIC 9(08). 03 FINDTM PIC 9(08). 03 GETCNT PIC 9(08). 03 OUTCNT PIC 9(08). 03 OUTTM PIC 9(08). 03 SORTCNT PIC 9(08). 03 SORTTM PIC 9(08). 03 UPOTTM PIC 9(08). 03 AGRTM PIC 9(08). 03 TOTALTH PIC 9(08). 03 RSLTCNT PIC 9(08). 03 AA1 PIC 9(08). 03 AA2 PIC 9(08). 03 AA3 PIC 9(08). 03 AA4 PIC 9(08). 03 AA5 PIC 9(08). 03 AA6 PIC 9(08). 03 AA7 PIC 9(08). 03 AA8 PIC 9(08). 03 AA9 PIC 9(08). 03 AA10 PIC 9(08). </pre>					

パターン名	ENDPROC	d #	s-d #	gf #	DQG #	node #
	DML ブロック					
	<pre> ----- PERFORM GETTIME. MOVE CTIME TO T7. COMPUTE T2 = T7 - T0. COMPUTE TFND = T2 - TOUT - TSORT - TMERGE - GAGRTM - TD1. * DISPLAY "*****". DISPLAY " * FINAL ACCOUNT FOR COBOL DML PROCESS *". DISPLAY "*****". DISPLAY "THE CARDINALITY =" CTOTAL. DISPLAY "THE NUMB. OF LOCATED =" TCNT. DISPLAY "THE NUMB. OF ACCESED =" TGCNTD. DISPLAY "THE NUMB. OF OUTPUT =" TGCNT. DISPLAY "THE NUMB. OF SORT =" TSCNT. DISPLAY "THE TIME FOR LOCATING =" TFND. DISPLAY "THE TIME FOR OUTPUT =" TOUT. DISPLAY "THE TIME FOR SORTING =" TSORT. DISPLAY "THE TIME FOR MERGING =" TMERGE. DISPLAY "THE TIME FOR AGGREGAT =" GAGRTM. DISPLAY "THE TIME FOR UPDATE =" TD1. DISPLAY "THE TIME FOR TOTAL TIME =" T2. * * * OPEN I-II RELID-F. READ RELID-F AT END GO TO RENDD. DISPLAY "RELID =" RELID IN RELID-R. MOVE ENDCODE IN WK-RELID TO ENDCODE IN RELID-R. MOVE STCODE IN WK-RELID TO STCODE IN RELID-R. MOVE ERRCNT IN WK-RELID TO ERRCNT IN RELID-R. MOVE TCNT TO FINDCNT IN RELID-R. MOVE TFND TO FINDTM IN RELID-R. MOVE TGCNTD TO GETCNT IN RELID-R. MOVE TGCNT TO OUTCNT IN RELID-R. MOVE TOUT TO OUTTM IN RELID-R. MOVE TGCNT TO SORTCNT IN RELID-R. ADD TSORT , TMERGE TO SORTTM IN RELID-R. MOVE TD1 TO UPDTM IN RELID-R. MOVE GAGRTM TO AGRTM IN RELID-R. MOVE T2 TO TOTALTM IN RELID-R. MOVE CTOTAL TO RSLTCNT IN RELID-R. MOVE AA1 IN WK-RELID TO AA1 IN RELID-R. MOVE AA2 IN WK-RELID TO AA2 IN RELID-R. MOVE AA3 IN WK-RELID TO AA3 IN RELID-R. MOVE AA4 IN WK-RELID TO AA4 IN RELID-R. MOVE AA5 IN WK-RELID TO AA5 IN RELID-R. MOVE AA6 IN WK-RELID TO AA6 IN RELID-R. MOVE AA7 IN WK-RELID TO AA7 IN RELID-R. MOVE AA8 IN WK-RELID TO AA8 IN RELID-R. MOVE AA9 IN WK-RELID TO AA9 IN RELID-R. MOVE AA10 IN WK-RELID TO AA10 IN RELID-R. REWRITE RELID-R. RENDD. CLOSE RELID-F. </pre>					

パターン名	MMUIDTWS	d #	s-d #	g f #	DQG #	node #
	DML ブロック					
* THE NUMB. OF ACCESSED OCCURRENCES						
01	CNT	PIC	9(08)	VALUE	0.	
01	GCNT	PIC	9(08)	VALUE	0.	
01	GCNTII	PIC	9(08)	VALUE	0.	
01	C1	PIC	9(08)	VALUE	0.	
01	G1	PIC	9(08)	VALUE	0.	
01	G01	PIC	9(08)	VALUE	0.	
01	S01	PIC	9(08)	VALUE	0.	
01	SCNT	PIC	9(08)	VALUE	0.	
01	TSCNT	PIC	9(08)	VALUE	0.	
01	TCNT	PIC	9(08)	VALUE	0.	
01	TGCNT	PIC	9(08)	VALUE	0.	
01	TGCNTII	PIC	9(08)	VALUE	0.	
* VARIABLES FOR STORING ELAPSE TIME.						
01	ONE	PIC	9	VALUE	1.	
01	T1	PIC	9(10)	VALUE	0.	
01	T2	PIC	9(10)	VALUE	0.	
01	T3	PIC	9(10)	VALUE	0.	
01	T4	PIC	9(10)	VALUE	0.	
01	T5	PIC	9(10)	VALUE	0.	
01	T0	PIC	9(10)	VALUE	0.	
01	T6	PIC	9(10)	VALUE	0.	
01	T7	PIC	9(10)	VALUE	0.	
01	T12	PIC	9(10)	VALUE	0.	
01	T13	PIC	9(10)	VALUE	0.	
01	T15	PIC	9(10)	VALUE	0.	
01	T16	PIC	9(10)	VALUE	0.	
01	T20	PIC	9(10)	VALUE	0.	
01	T31	PIC	9(10)	VALUE	0.	
01	T51	PIC	9(10)	VALUE	0.	
01	SRT1	PIC	9(10)	VALUE	0.	
01	SRT2	PIC	9(10)	VALUE	0.	
01	SRTTM	PIC	9(10)	VALUE	0.	
01	GSRTTM	PIC	9(10)	VALUE	0.	
01	AGT1	PIC	9(10)	VALUE	0.	
01	AGT2	PIC	9(10)	VALUE	0.	
01	AGRTMO	PIC	9(10)	VALUE	0.	
01	TFND	PIC	9(10)	VALUE	0.	
01	TOUT	PIC	9(10)	VALUE	0.	
01	TSURT	PIC	9(10)	VALUE	0.	
01	TMERGE	PIC	9(10)	VALUE	0.	
01	GAGRTM	PIC	9(10)	VALUE	0.	
* VARIABLE FOR STORING THE ELAPSE TIME OF UPDATING.						
01	TD0	PIC	9(10)	VALUE	0.	
01	TD1	PIC	9(10)	VALUE	0.	
01	TIME1	PIC	9(08)			
01	TIME2	REDEFINES TIME1.				
01	03 TM	PIC	9(02)	OCCURS	4.	
01	CTIME	PIC	9(10)			
* CONSTANT						
01	FALSE	PIC	9(02)	VALUE	0.	
01	TRUE	PIC	9(02)	VALUE	1.	
01	ENDD	PIC	9(02)	VALUE	0.	
01	ENDF	PIC	9(02)	VALUE	0.	
01	YES	PIC	9(02)	VALUE	1.	
01	NOO	PIC	9(02)	VALUE	0.	
01	ONN	PIC	9(02)	VALUE	1.	
01	OFFF	PIC	9(02)	VALUE	0.	

パターン名	MMUIDTWS	d #	s-d #	gf #	DQG #	node #
	DML ブロック					
<pre> * * * WORKING VARIABLES 01 I PIC S9(08). 01 J PIC S9(08). 01 Q PIC S9(08). 01 UI PIC S9(08). 01 QJ PIC S9(08). 01 H PIC S9(08). 01 K PIC S9(08). 01 L PIC S9(08). 01 M PIC S9(08). 01 N PIC S9(08). 01 II PIC S9(08). 01 JJ PIC S9(08). 01 JH PIC S9(08). 01 KK PIC S9(08). 01 PT PIC 9(08) VALUE 0. 01 PPT PIC 9(08) VALUE 0. 01 PTNO PIC 9(08) VALUE 0. 01 PPTNO PIC 9(08) VALUE 0. 01 LPTNO PIC 9(08) VALUE 0. 01 NRPTNO PIC 9(08) VALUE 0. 01 NLPTNO PIC 9(08) VALUE 0. 01 RPPT PIC 9(08) VALUE 0. 01 RPPTNO PIC 9(08) VALUE 0. 01 NPPTNO PIC 9(08) VALUE 0. 01 PTN PIC 9(08) VALUE 0. 01 LPT PIC 9(08) VALUE 0. * NAME OF RECORD-TYPE (REN) AND ITEM (ITN) 01 REN PIC X(16). 01 ITN PIC X(16). 01 ERR1 PIC 9(03). 01 ERR2 PIC 9(02). 01 WST PIC X(02). 01 CTOTAL PIC 9(04) VALUE 0. 01 CNTDQG PIC 9(08). 01 MODES PIC 9. 01 MODEE PIC 9(05). * WORK AREA FOR ENDCODE ETC (RESULTID-F) 01 WK-RELID. 03 ENDCODE PIC 9(04) VALUE 0. 03 STCODE PIC 9(08) VALUE 0. 03 ERRCNT PIC 9(08) VALUE 0. 03 AA1 PIC 9(08) VALUE 0. 03 AA2 PIC 9(08) VALUE 0. 03 AA3 PIC 9(03) VALUE 0. 03 AA4 PIC 9(08) VALUE 0. 03 AA5 PIC 9(08) VALUE 0. 03 AA6 PIC 9(08) VALUE 0. 03 AA7 PIC 9(08) VALUE 0. 03 AA8 PIC 9(03) VALUE 0. 03 AA9 PIC 9(08) VALUE 0. 03 AA10 PIC 9(03) VALUE 0. </pre>						

パターン名	MMUISUB	d #	s-d #	gf #	DQG #	node #
	DML ブロック					
<pre> *----- * ----- FRTRN (FAILURE RETURN) ----- * * PT = A POINTER OF THE CURRENT NODE * PPT = A POINTER OF THE FAILURE RETURNED NODE. * * FRTRN SECTION. * PERFORM LRM. * FRTRN1. * IF NRPPTND > NLPTND GO TO FRTRN5. * MOVE LENGTT IN SRSLT (NLPTND) TO K. * MOVE CVPT IN SRSLT (NLPTND) TO H. * IF H > 0 * SUBTRACT K FROM H. * * MOVE NRPPTND TO I. * FRTRN2. * IF I > NLPTND GO TO FRTRN3. * MOVE H TO CVPT IN SRSLT (I). * IF I NOT = NRPPTND * MOVE 0 TO LENGTT IN SRSLT (I). * ADD 1 TO I. * GO TO FRTRN2. * FRTRN3. * IF NLPTND NOT < CSRT GO TO FRTRN5. * ADD 1, NLPTND GIVING J. * FRTRN4. * IF J > CSRT GO TO FRTRN5. * IF LENGTT IN SRSLT (J) > 0 * SUBTRACT K FROM LENGTT IN SRSLT (J). * ADD 1 TO J. * GO TO FRTRN4. * FRTRN5. * EXIT. *----- * ----- RESULT ----- * * PT = A POINTER OF THE OUTPUT NODE * DBK = DATABASE-KEY * * RESULT SECTION. * MOVE 1 TO I. * RESULT00. * IF I NOT > CSRT * IF IPT IN SRSLT (I) = PT * GO TO RESULT01 * ELSE ADD 1 TO I GO TO RESULT00 * ELSE GO TO RESULT5. * RESULT01. * IF I = 1 MOVE 1 TO LENGTT IN SRSLT (I). * MOVE I TO NPTND. * ADD 1, CVPT IN SRSLT (I) GIVING J. * ADD CVPT IN SRSLT (I), LENGTT IN SRSLT (I) * GIVING JJ. * RESULT1. * IF J NOT > JJ </pre>						

パターン名	MMUISUB	d #	s-d #	gf #	DQG #	node #
	DML ブロック					
	<pre> MOVE DBK TO DBKEY (I, J) ADD 1 TO J GO TO RESULT1. ADD LENGTT IN SRSLT (I) TO CVPT IN SRSLT (I). * IF NPTNO < CSRT ADD 1, NPTNO GIVING K ADD LENGTT IN SRSLT (NPTNO) TO LENGTT IN SRSLT (K). * IF NPTNO NOT > 1 GO TO RESULT5. IF CVPT IN SRSLT (I) NOT < CVPT IN SRSLT (NPTNO) GO TO RESULT5. MOVE LENGTT IN SRSLT (NPTNO) TO K. COMPUTE J = CVPT IN SRSLT (NPTNO) LENGTT IN SRSLT (NPTNO). MOVE 1 TO I. COMPUTE II = NPTNO - 1. RESULT2. IF I > II GO TO RESULT5. MOVE 1 TO H. RESULT3. IF H > K GO TO RESULT4. ADD J H GIVING JH. MOVE DBKEY (I, H) TO DBKEY (I, JH). ADD 1 TO H. GO TO RESULT3. RESULT4. MOVE CVPT IN SRSLT (NPTNO) TO CVPT IN SRSLT (I). ADD 1 TO I. GO TO RESULT2. RESULT5. EXIT. *-----* * LRM LRM SECTION. MOVE 0 TO LPTNO. COMPUTE RPPTNO = CNDT + 1; MOVE 0 TO K. MOVE 1 TO I. LRM1. IF I > CNDT GO TO LRM-EXIT. IF NPT IN NDTRL (I) NOT = PT GO TO LRM3. MOVE ND-ND IN NDTRL (I) TO PTN PINO. IF LPT-ND IN NDTRL (I) NOT = 0 GO TO LRM3. ---- LPT-ND (I) = 0 ---- MOVE 1 TO J. LRM2. IF J > CSRT GO TO LRM2. IF ND-ND IN SRSLT (J) > PTN GO TO LRM2. MOVE NPT IN SRSLT (J) TO LPT. MOVE ND-ND IN SRSLT (J) TO LPTNO. IF ND-ND IN SRSLT (J) = PTN MOVE J TO NPTNO GO TO LRM21. ADD 1 TO J. GO TO LRM2. </pre>					

パターン名	MMU1SUB	d #	s-d #	gf #	DQG #	node #
	DML ブロック					
	<pre> LM2. COMPUTE NLPTNO = J - 1. LM21. MOVE LPTNO TO LPT-NO IN NDABL (1). MOVE NLPTNO TO SNLPTNO IN NDABL (1). GO TO LRM4. * ---- LPT-NO (1) > 0 ---- LRM3. MOVE LPT-NO IN NDABL (1) TO LPTNO. MOVE NPT IN NDABL (LPTNO) TO LPT. MOVE SNLPTNO IN NDABL (1) TO NLPTNO. LRM4. IF K = 1 GO TO LRM-EXIT. MOVE 1 TO K. * LRM5. IF NPT IN NDABL (1) NOT = PPT GO TO LRM9. MOVE ND-NO IN NDABL (1) TO PTN PPINO. IF RPT-NO IN NDABL (1) NOT = 0 GO TO LRM7. * ---- RPT-NO (1) = 0 ---- MOVE CSRT TO J. LRM6. IF J < 1 GO TO LM3. IF ND-NO IN SRSLT (J) < PTN GO TO LM3. MOVE NPT IN SRSLT (J) TO RPPT. MOVE ND-NO IN SRSLT (J) TO RRPPTNO. IF ND-NO IN SRSLT (J) = PTN MOVE J TO NRPPTNO GO TO LM31. SUBTRACT 1 FROM J. GO TO LRM6. LM3. COMPUTE NRPPTNO = J + 1. LM31. MOVE RRPPTNO TO RPT-NO IN NDABL (1). MOVE NRPPTNO TO SRPPTNO IN NDABL (1). GO TO LRM8. * ---- RPT-NP (1) > 0 ---- LRM7. MOVE RPT-NO IN NDABL (1) TO RRPPTNO. MOVE NPT IN NDABL (RPPTNO) TO RPPT. MOVE SRPPTNO IN NDABL (1) TO NRPPTNO. LRM8. IF K = 1 GO TO LRM-EXIT. MOVE 1 TO K. LRM9. ADD 1 TO I. GO TO LRM1. LRM-ERR. DISPLAY "LRM ERROR PT =" PT ", PPT =" PPT. LRM-EXIT. EXIT. *----- ERRP SECTION. ERRP1. DISPLAY "PATTERN NUMB. = " ERR1 ", ERROR CODE = " ERR2. STOP RUN. </pre>					

パターン名	MMUISUB	d #	s-d #	gf #	DQG #	node #
	DML ブロック					
	<pre> ERR-EXIT. EXIT. ----- * SUBROUTINE SUBRTNS SECTION. * CTIME51 * CTIME51 T51 = ACCUMULETED TIME. * CTIME51. PERFORM GETTIME. MOVE CTIME TO T5. COMPUTE T15 = (T5 - T4). CTIME510. EXIT. * CTIME31 * CTIME31 * CTIME31. PERFORM GETTIME. MOVE CTIME TO T3. COMPUTE T13 = (T3 - T2). CTIME310. EXIT. * * COUNT * COUNT. ADD 1 TO CNT. COUNTD. EXIT. * * GCOUNT * GCOUNT. ADD 1 TO GCNTU. GCOUNTD. EXIT. * * OUTPUT COUNT. * GCOUNTP. ADD 1 TO GCNT. GCOUNTPD. EXIT. * * GETTIME * GETTIME. ACCEPT TIME1 FROM TIME. COMPUTE CTIME = TM (1) * 360000 + TM (2) * 6000 + TM (3) * 100 + TM (4). GETT-E. EXIT. * * PR-ACCU1 * PR-ACCU1. DISPLAY "DQG#" I. DISPLAY " THE CARDINALITY = " CNTDQG. DISPLAY " THE NUMB. OF LOCATED OCC = " CNT. DISPLAY " THE NUMB. OF ACCESSED OCC = " GCNTU. DISPLAY " THE NUMB. OF OUTPUT OCC = " GCNT. DISPLAY " THE NUMB. OF SORTED OCC = " SCNT. DISPLAY " THE TIME FOR LOCATING = " T13. DISPLAY " THE TIME FOR OUTPUT = " T15. DISPLAY " THE TIME FOR SORTING = " SKT13. PR-ACCEXIT. EXIT. </pre>					

付記5 集積関数処理の評価

A. 評価目的

LDP-V2に対して追加された以下の機能に対するLDPの性能を評価する。

- 集積関数
- 結果の関係作業領域(RWS)への格納
- 結果の編集出力処理

i) 集積関数

集積関数进行处理する為に変更されたLDPの性能評価として、各構成モジュールの経過時間を測定し、隘路となるモジュールを明らかにする。

又、LDPから生成されるCOBOL DMLに対しても各処理モジュールの経過時間を測定し、隘路となる処理モジュールを明らかにする。

ii) 結果の関係作業領域(RWS)への格納

検索結果を関係作業領域(RWS)に格納する処理モジュール(LOAD WS)の性能評価として、経過時間を測定する。

iii) 結果の編集出力処理

検索結果は関係作業領域(RWS)からアクセスされ、編集して端末画面に表示される。この時、RWSからアクセスされた結果は、いったんファイルに出力され、分類され冗長な組を除き、編集処理されて画面に表示される。結果の編集出力処理の性能評価として、RWSのアクセスとファイルの分類の経過時間を測定する。

B. 方針

評価の方針は、以下の様である。

- a) 単一利用者のもとで、LDP-V3を実行する。他の利用者ジョブによる影響の無い状態での性能を評価する。
- b) LDP-V3の各モジュールに対しては、PL/Iが提供する時間取得(e. g. TM=TIME)、COBOL DMLプログラムの各処理サブルーチンに対してはCOBOLが提供する時間取得(e. g. ACCEPT TM FROM 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回行う。M-170Fで経過時間を単一利用者下で測定しても、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-V3は、M-170Fで実行する。
- e) 問合せで検索するデータベースとしては、DCDBSを用いる。

C. LDPの性能評価

LDPを構成する各モジュールのCPU負荷として経過時間を測定し、隘路となるモジュールを明らかにする。LDPを構成するモジュールは、図5.1の様である。

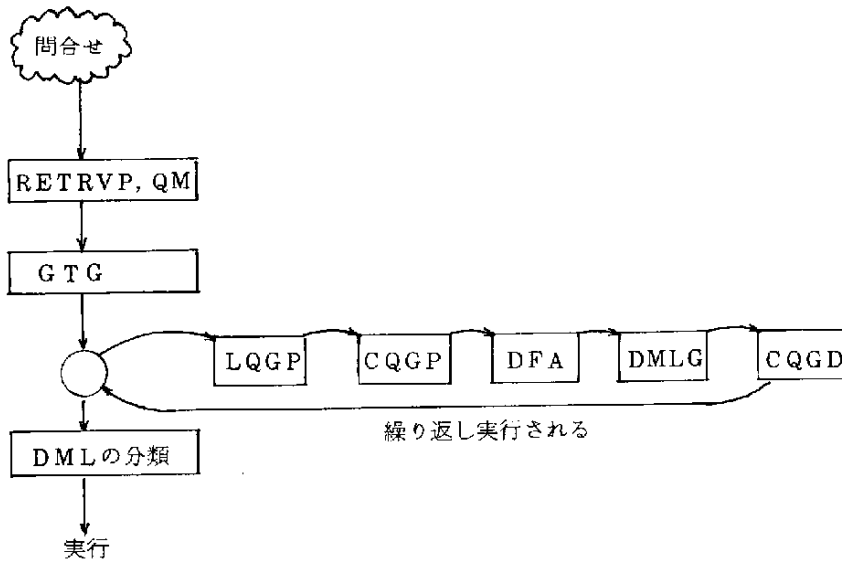


図5.1 LDP-V3のモジュール構成

各モジュールLQGP、CQGP、DFA、DMLG、CQGDは各々繰り返し実行される。各経過時間を各々総計したものを各モジュールの経過時間とする。

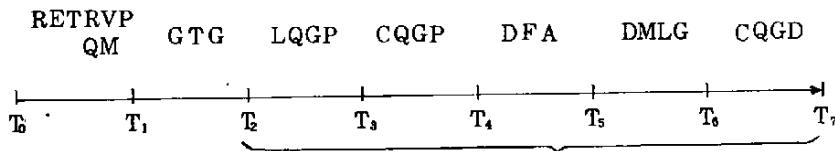


図5.2 LDPの測定時間

各モジュールは、次の様な処理を行う。(i)RETRVP、QMは、RQLによって記述された問合せの文字列からQ-treeを生成する。(ii)GTGは、Q-treeから集積関数処理するG-treeを生成する。(iii)LQGPは、treeからLCS問合せグラフ(LQG)を生成する。(iv)CQGPは、LQGからCODASYL問合せグラフ(CQG)を生成する。(v)DFAは、CQGからアクセスパスを生成する。(vi)DMLGは、アクセスパスからCOBOL DMLを生成する。(vii)CQGDは、CQGを連結可能なCQGiに分割し、スケジューリングを行う。

```

RANGE (D,DOCUMENT);
RETRIEVE INTO R1 (D.FISCAL-YEAR,
    MAX=COST=MAX(D.COST BY D.FISCAL-YEAR));

RANGE (D,DOCUMENT);
RETRIEVE INTO R2 (D.FISCAL-YEAR,
    MIN=COST=MIN(D.COST BY D.FISCAL-YEAR),
    MAX=COST=MAX(D.COST BY D.FISCAL-YEAR));

RANGE (D,DOCUMENT);
RETRIEVE INTO R3 (D.FISCAL-YEAR,
    MIN=COST=MIN(D.COST BY D.FISCAL-YEAR),
    AVR=COST=AVER(D.COST, D.@D BY D.FISCAL-YEAR),
    MAX=COST=MAX(D.COST BY D.FISCAL-YEAR));

RANGE (D,DOCUMENT);
RETRIEVE INTO R4 (D.FISCAL-YEAR,
    MIN=COST=MIN(D.COST BY D.FISCAL-YEAR),
    AVR=COST=AVER(D.COST, D.@D BY D.FISCAL-YEAR),
    SUM=COST=SUM(D.COST, D.@D BY D.FISCAL-YEAR),
    MAX=COST=MAX(D.COST BY D.FISCAL-YEAR));

RANGE (D,DOCUMENT);
RETRIEVE INTO R5 (D.FISCAL-YEAR,
    MIN=COST=MIN(D.COST BY D.FISCAL-YEAR),
    AVR=COST=AVER(D.COST, D.@D BY D.FISCAL-YEAR),
    SUM=COST=SUM(D.COST, D.@D BY D.FISCAL-YEAR),
    DCNT=COUNT(D.@D BY D.FISCAL-YEAR),
    MAX=COST=MAX(D.COST BY D.FISCAL-YEAR));

```

図5.3 評価用問合せ

各モジュールの経過時間

問合せとして、group-by 集積関数の異なる図5.3の様な問合せを実行し、各モジュールの経過時間を測定した。各問合せに対する各モジュールの経過時間は図5.4の様である。

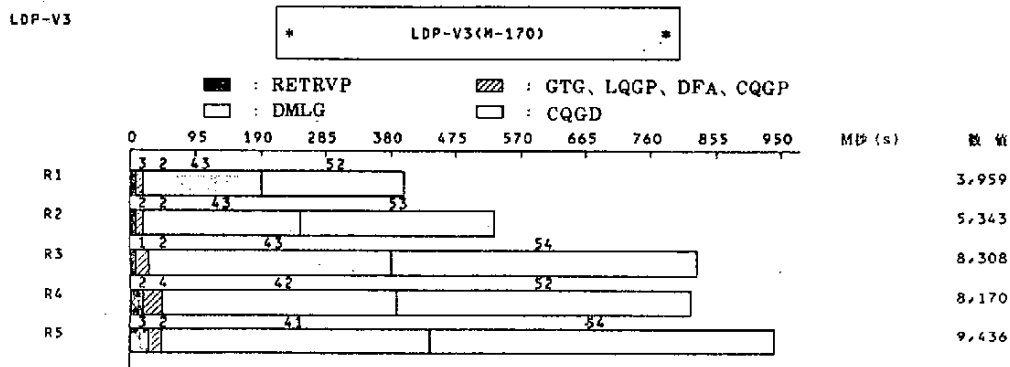


図5.4 各処理モジュールの経過時間

図から分る様に、モジュールDMLGとCQGDの経過時間が、各々全体の41～43%、52～

54%を占めている。

DMLGは、LDP-V1.5の評価では全体に占める割合が70～90%であった。この為、改良を図り経過時間の短縮を実現している。今回の評価では、DMLパターンの一部をコピーライブラリに登録し、生成するステートメント数の減少を行った。LDP-V1.5で使用した問合せをLDP-V3で実行した結果とLDP-V1.5の改良版の結果を各々、図5.5、図5.6に示す。

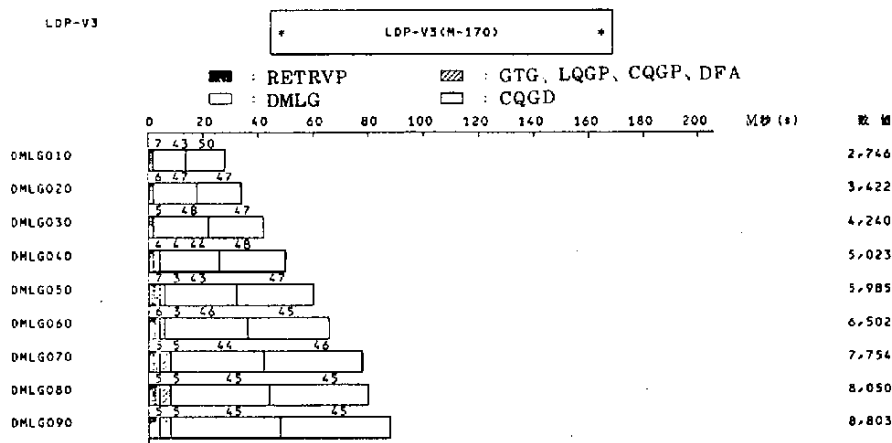


図5.5 LDP-V3の実行結果

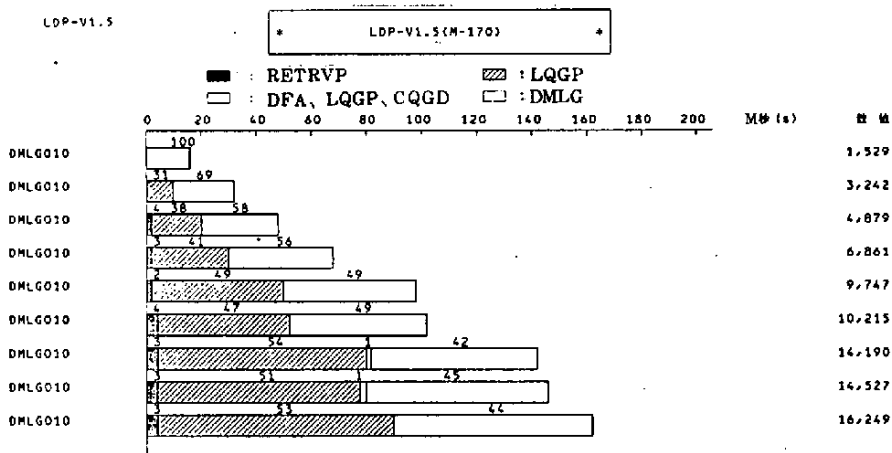


図5.6 LDP-V1.5の改良版の実行結果

LDP-V1.5の改良版と比べると、問合せの最初の方(DMLG010、020)では経過時間がよけいにかかっている。しかし、問合せの番号が大きい方(番号が大きくなるにつれてアクセス木のノード数が増加する)では、逆に大巾に経過時間が減少している。これは、LDP-V1.5でネックとなっていたDMLGに対する改良の効果と考えられる。

・DMLパターンのCOPY LIBRARY化

・繰り返しパターンの導入

これにより、DMLの生成時の入出力ステートメント数が減少した為と考えられる。

CQGDでは、CQGを入力して、連結な部分グラフCQGi ($i = 1, 2, \dots, n$)に分割し、その中間結果の結合、又は直積のスケジュールを決定する。集積関数进行处理する為、中間結果の結合などのスケジュールの決定の負荷が大きくなっている。

D COBOL DMLの性能評価

LDPから生成されたCOBOL DMLプログラムを次の様な処理モジュールに分け、各処理モジュールの経過時間を測定し、COBOL DML実行時の隘路となる処理モジュールを明らかにする。

処理モジュール

- i) 実現値のアクセス処理
- ii) 中間結果、結果の出力処理
- iii) 中間結果、結果の分類処理
- iv) 中間結果の結合処理
- v) aggregate 処理

これらの処理モジュールは、繰り返し実行されるので、各経過時間の総和をその処理モジュールの経過時間とする。

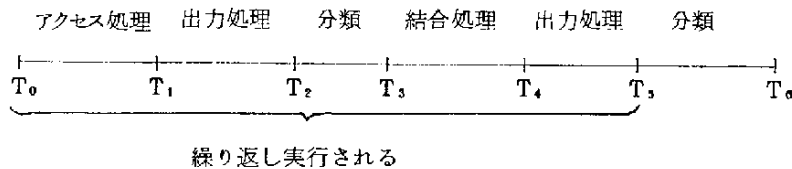


図5.7 COBOL DMLの測定時間

評価問合せを図5.3に示す。測定結果は、表5.1の様である。各処理モジュールの全体に占める経過時間は図5.8の様である。全体の経過時間に対して、分類処理の占める割合は約60%となっている。又、結合処理とアクセス処理が各々15~20%を占めている。この問合せの場合、中間結果として得られる件数が少ない(9件である)ため、結合処理の占める割合は余り大きくない。中間結果数が増えると、全体に占める各モジュールの経過時間が大きく変化する。

表5.1 測定結果

問合せ 処理単位	R 1	R 2	R 3	R 4	R 5
実現値のアクセス 〔10msec〕	5 5 0	7 9 0	1 0 4 8	1 2 9 5	1 5 3 3
中間結果、結果の出力 〔10msec〕	1	2	0	1	2
中間結果、結果の分類 〔10msec〕	2 2 3 7	3 2 4 5	4 2 9 0	5 3 6 6	6 3 5 2
中間結果の結合 〔10msec〕	6 6 4	9 2 8	1 3 3 8	1 7 5 9	1 9 4 3
aggregate 処理 〔10msec〕	1 0 8	2 3 9	4 0 3	5 3 6	7 1 8
計 〔10msec〕	3 5 6 0	5 2 0 4	7 0 7 9	8 9 5 9	1 0 5 4 8

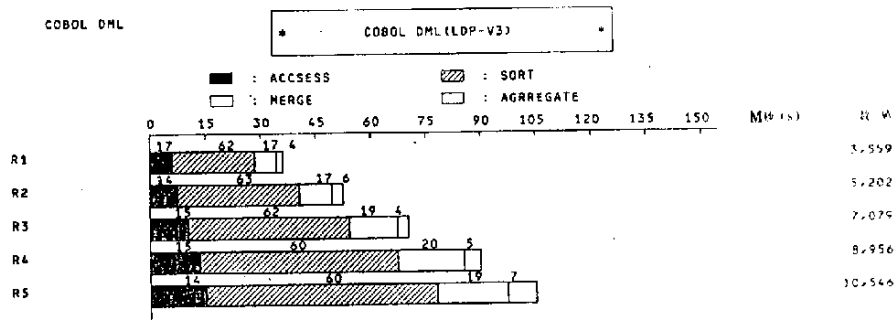


図5.8 各処理モジュールの経過時間

```

RANGE (D,DOCUMENT) (PD,P-D) (P,PROJECT);
RETRIEVE INTO R11 (D,JND)
WHERE
  ANY (P,EP BY D,JND WHERE D,ED = PD,ED AND PD,EP = P,EP) = 0;
=====
RANGE (D,DOCUMENT) (PD,P-D) (P,PROJECT);
RETRIEVE INTO R12 (D,JND)
WHERE
  COUNT (P,EP BY D,JND WHERE D,ED = PD,ED AND PD,EP = P,EP) = 0;
=====
RANGE (D,DOCUMENT) (PD,P-D) (P,PROJECT);
RETRIEVE INTO R13 (D,JND,
  PROJ-NO=COUNT (P,EP BY D,JND
    WHERE D,ED = PD,ED AND PD,EP = P,EP),
  PROJ-EXIST=ANY (P,EP BY D,JND
    WHERE D,ED = PD,ED AND PD,EP = P,EP));
=====
RANGE (D,DOCUMENT);
RETRIEVE INTO R14 (PC=D,P-COMPANY,
  COST=AVER (D,COST, ED BY D,P-COMPANY))
WHERE
  AVER (D,COST, D,ED) LE AVER (D,COST, D,ED BY D,P-COMPANY);

```

図5.9 中間結果数の多い問合せ

図 5.9 の問合せでは、group-by 属性として JNO、あるいは P-COMPANY が指定されている。集積処理の結果数は、JNO が 270、P-COMPANY が 56 である。中間結果数が多い場合の測定結果を表 5.2 に示す。各処理モジュールの経過時間の全体に占める割合を図 5.10 に示す。図から分るように、結合処理は全体の経過時間に占める割合が 60～90% である。又、全体の経過時間も大巾に増加する。これは、中間結果の結合処理で、中間結果の 1 つの組に対して、他の全ての中間結果の組と結合処理を行うためである。

表 5.2 測定結果

問合せ 処理単位	R11	R12	R13	R14
実現値のアクセス 〔10msec〕	596	599	864	989
中間結果、結果の出力 〔10msec〕	1	0	2	2
中間結果、結果の分類 〔10msec〕	2477	2471	3683	4464
中間結果の結合 〔10msec〕	19398	19402	64424	10115
aggregate 処理 〔10msec〕	390	390	685	481
計〔10msec〕	22862	22862	69658	16051

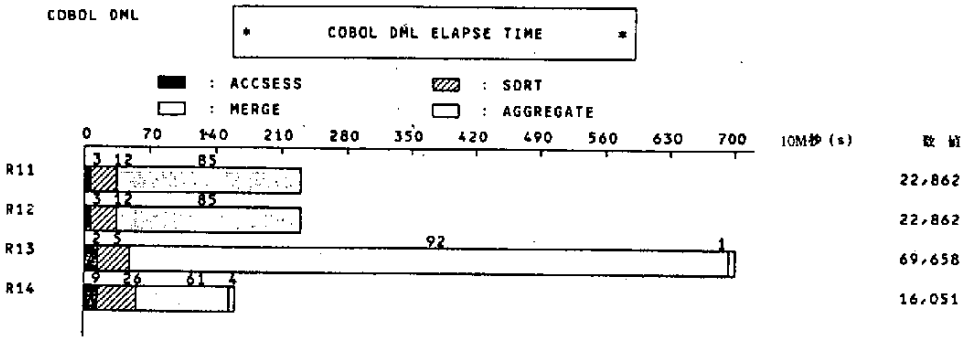


図 5.10 各処理モジュールの経過時間

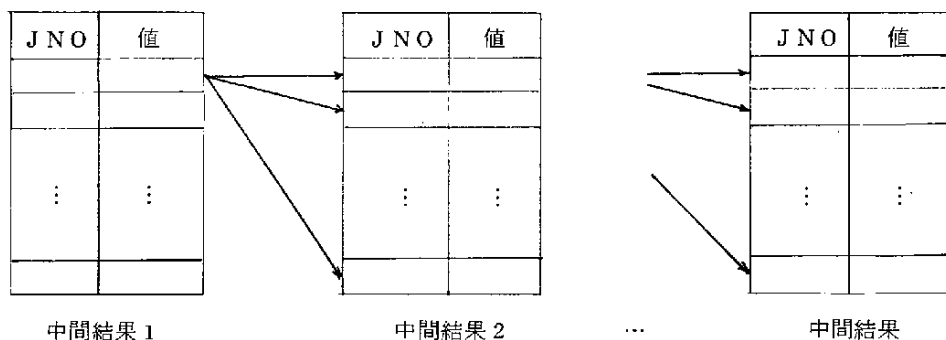


図 5.1 1 中間結果の結合処理

結合処理は、次の様に処理される。

(1) 最初の間結果から組を 1 つ読み出す。組がなければ終了。

(2) 次の中間結果を読み、結合条件を満たす組を見つける。

見つかったら、次の中間結果に対して(2)の処理を続ける。最終中間結果で結合条件が満足されたら、結果として出力する。

見つからなかったら、(1)へもどる。

LDP-V3 から生成される COBOL DML プログラムの経過時間を短縮する為には、ムダな中間結果の read をなくすことが必要となる。結合条件によるスケジューリングと結合属性の分類による処理の効率化を検討する。

E LOADWS の CPU 負荷

LOADWS は、問合せの結果を関係作業領域 (RWS) に格納する。又、問合せの実行時にアクセスしたセットの結合度、更新によるレコード実現値数の増減情報で HI 情報の更新を行う。

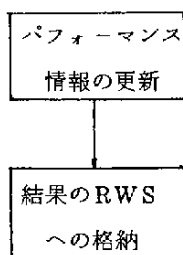


図 5.1 2 LOADWS の処理概要

RWS は、図 5.1 3 の様な構造で、AIM の下で管理される。

結果の属性が数字データ、あるいは文字データにより、次の様に格納方法が異なる。

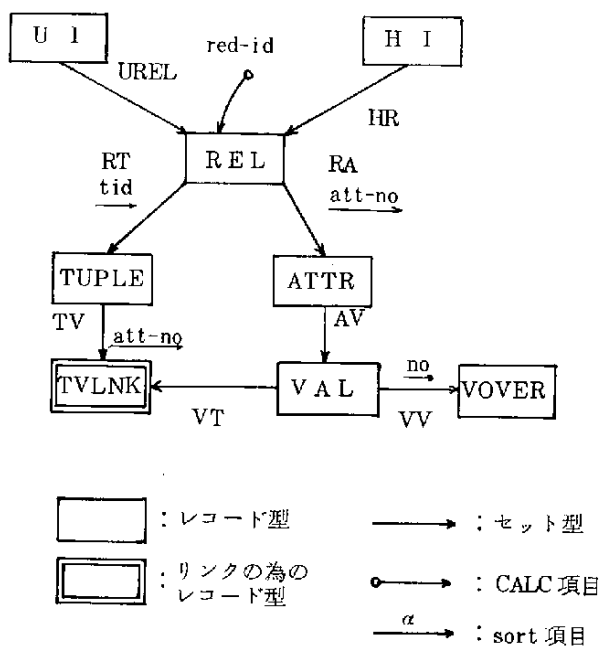
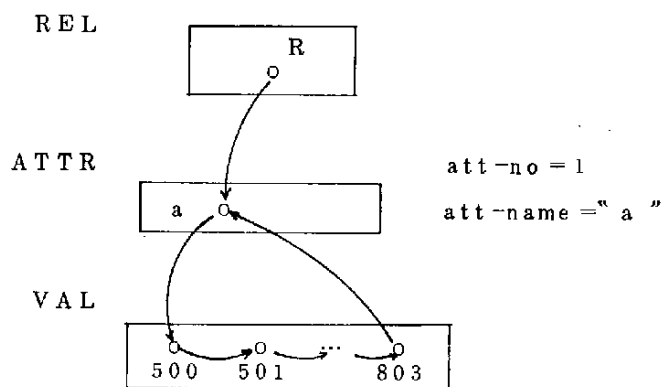


図5.13 RWSの構造

数字データの格納方法

数字データは、バイナリ変換して文字タイプ4桁の領域に格納する。

例えば、 $R[a] = \{500, 501, \dots, 803\}$ 、 $att-no(a) = 1$ の場合、格納は次の様である。



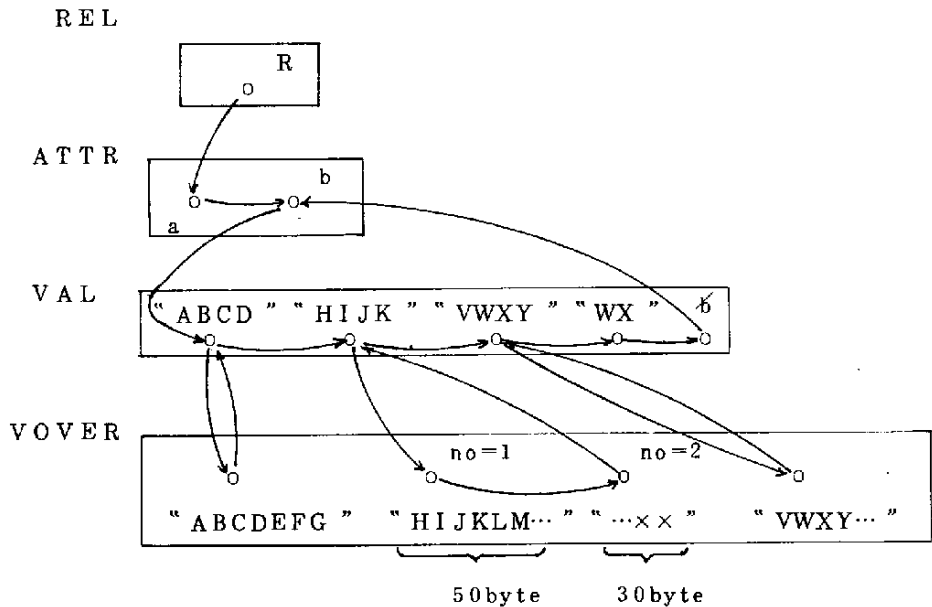
文字データの格納方法

文字データは、次の様に格納される。

- i) 文字データの時、**VAL**には各値の先頭4 byteが格納される。
- ii) 完全なデータは、**VOVER**に50 byte単位に格納される。
- iii) 4文字以内のデータは、**VAL**だけに格納される。

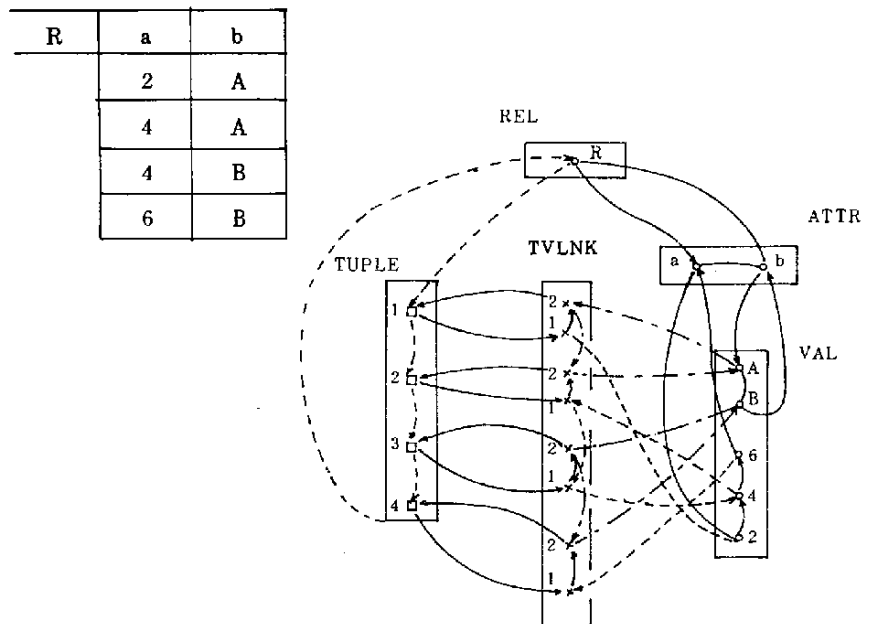
iv) ブランクは、iii)と同様に扱われる。

例えば、 $R(b) = \{ \text{" ABCDEFG "}, \text{" HIJKLM..... \times \times "}, \text{" VWXYZ 1 2 3 4 5 6 7 8 9 ABCD "}, \text{" WX "}, \text{" } \}$, $\text{att-no}(b) = 2$ の場合、格納は次の様である。



又、結果の各組は、属性が取る値に対してリンク (TVLNK) が結ばれている。

例えば、結果関係が次の様な場合、RWSへの格納は図の様である。



同一問合せに対して、得られる結果数を変えて（制限条件を変更）、LOADWSのCPU負荷（経過時間）を測定した。

問合せとしては、以下の問合せを実行した。

```
RANGE (D,DOCUMENT) (DA,DOC-AUTH) (A,AUTHOR);
RETRIEVE INTO RREL01 (U,JND,NOA=DA.AUTH-NO,
LN=A.LAST-NAME, FN=A.FIRST-NAME)
WHERE
D.ED = DA.ED AND DA.EA = A.EA AND D.JND LE 1780;
```

制限条件の値を変更して
結果の実現値を調整

結果のスキーマは、次の通りである。

```
(REL-NAME=RREL01, DEGREE= 4, CARDINALITY= 20, WIDTH= 69,
IPMODE=1, FLAG=0, EXIST=1, ENDCODE=0000)
```

1AT#1ATT-NAME	1TYP1	1LNTH1	1CARD1	1TYPI	1DOM1
1 JND	1	7	151	01JND	
2 INOA	1	2	21	01NOA	
3 ILN	1	30	161	01ILN	
4 IFN	1	30	121	01IFN	

問合せの実行結果を表 5.3 に示す。各問合せの経過時間は図 5.1 4 の様である。LOADWSの経過時間は、格納する実現数に対してほぼ単調に増加していることが分る。実現値数が300を超えるあたりでは、結果を格納するのに10数分も必要としている。これは、結果属性の値が、結果の組と属性に対してインバテッドリスト構造を取っている為と考えられる。LOADWSは、問合せから生成されるCOBOL DMLプログラムの実行ステップ（プリコンパイル～コンパイル～リンク～実行～結果格納）内の隘路となる。結果の格納処理を効率的に行う為には、組をそのまま格納することを検討する必要があると考えている。

表 5.3 LOADWS の Cpu 負荷

目標集合 の組件数	JNO の件数	NOA の件数	L N の件数	F N の件数	LOADWS の負荷 [msec]	[sec]
20	15	2	16	12	31965	31s
40	28	4	35	16	58631	58s
61	43	4	53	18	89146	1 _M 29s
82	57	4	74	19	125282	2 _M 5s
99	68	4	89	19	162708	2 _M 42s
121	73	9	105	20	197655	3 _M 17s
142	78	9	113	20	229020	3 _M 49s
161	83	10	118	20	220719	3 _M 40s
181	94	10	136	21	275161	4 _M 35s
200	110	10	149	22	373902	6 _M 13s
218	123	10	163	23	384472	6 _M 24s
238	139	10	179	24	413877	6 _M 53s
258	154	10	197	24	460869	7 _M 40s
279	169	10	208	24	562048	9 _M 22s
298	179	10	225	24	599458	9 _M 59s
320	194	10	240	24	696979	11 _M 36s

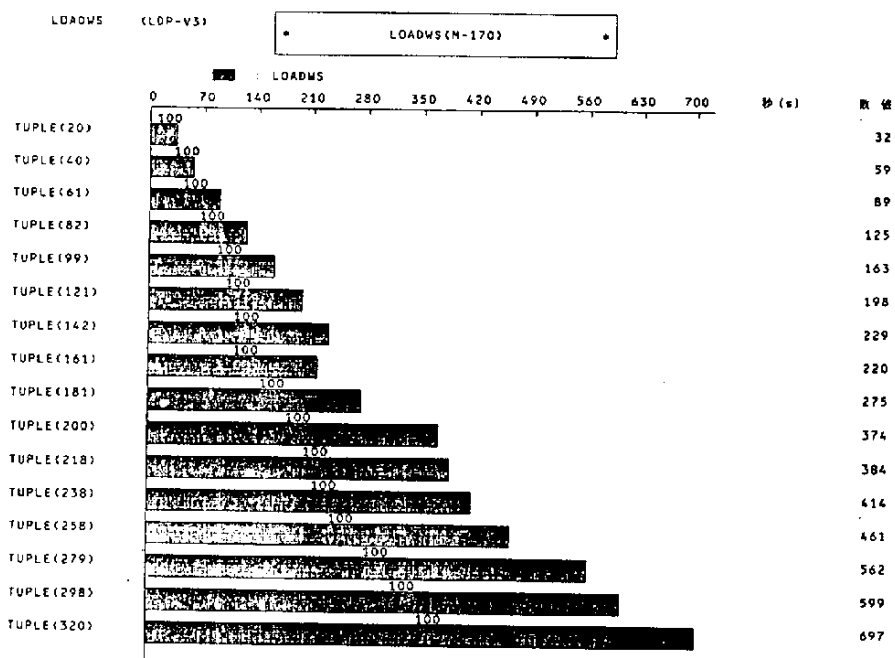


図5.14 LOADWSの経過時間

F 結果の出力処理の性能評価

問合せの検索結果は、関係作業領域 (RWS) からアクセスされ、編集されて端末画面に表示される。

結果の出力処理は、図 5.1 5 の様である。

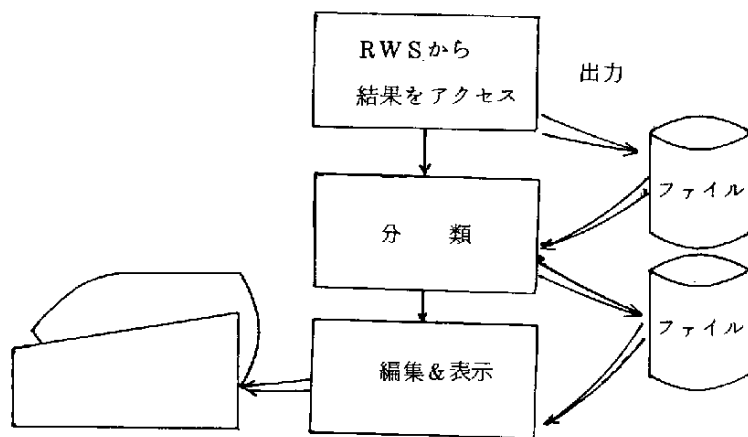


図5.15 結果の出力処理

結果内から冗長な組を削除する為、いったんファイルに出力し、分類を行う。結果の出力処理の性能評価として、RWSから結果をアクセスしファイルを作成する処理とファイルの分類処理の経過時間を測定した。

出力した結果関係に対する属性構成は次の様である。図 5.1 6は、結果の出力例である。

#00060 OUTP RREL01 (JNO/17, NDA/12, LN/C30, FN/C10);

** RREL01		**	
* RESULT CARDINALITY = 000000020			
IJNO	INDLN	IFN	I
I	712I 1IFELLER	IW.	I
I	1191I 1IKLEINROCK	IL.	I
I	1223I 1IKLIMBLE	IJ.	I
I	1223I 2IKOFFEMAN	IK.	I
I	1247I 1IDOUQUE	IB.	I
I	1247I 2INIJSSEN	IG.	I
I	1383I 1IKLEINROCK	IL.	I
I	1398I 1IHASSELMEIER	IH.	I
I	1398I 2ISPRUTH	IW.	I
I	1462I 1INIJSSEN	IG.	I
I	1538I 1IKERR	ID.	I
I	1614I 1IJARDINE	ID.	I
I	1615I 1IDATE	IC.	I
I	1621I 1IGHOSH	IS.	I
I	1625I 1INIJSSEN	IG.	I
I	1638I 1IAUSTIN	ID.	I
I	1727I 1ILOCKEMANN	IP.	I
I	1727I 2INEUHOLD	IE.	I
I	1780I 1IGOTTLIEB	IC.	I
I	1780I 2IGOTTLIEB	IL.	I

図5.1 6 結果の出力

表 5.4 に測定結果を示す。RWSのアクセスに対して、分類処理の経過時間がほぼ2倍になっている。図 5.1 7は、RWSのアクセス時間に分類の経過時間を加えた出力処理の経過時間（結果が表示されるまでの時間）を表わしている。実現値数が多い場合、端末に結果があらわれるまでかなりの時間を必要とする（例えば、実現値数200では、約40秒かかる）。

結果の出力処理の経過時間を短縮する為には、ファイルへの出力と分類を必要としない処理方法を検討する必要がある。

表5.4 測定結果

実現値数	結果のアクセス(msec)	分類(msec)
20	1494	3198
40	2509	7570
61	4166	9952
82	5741	12359
99	7093	18868
121	9187	22221
161	13052	23076
181	15260	26425
200	19068	28943
218	20185	30013
238	20001	40858
258	24780	35954
279	24531	38527
298	26200	41621
320	30877	44200

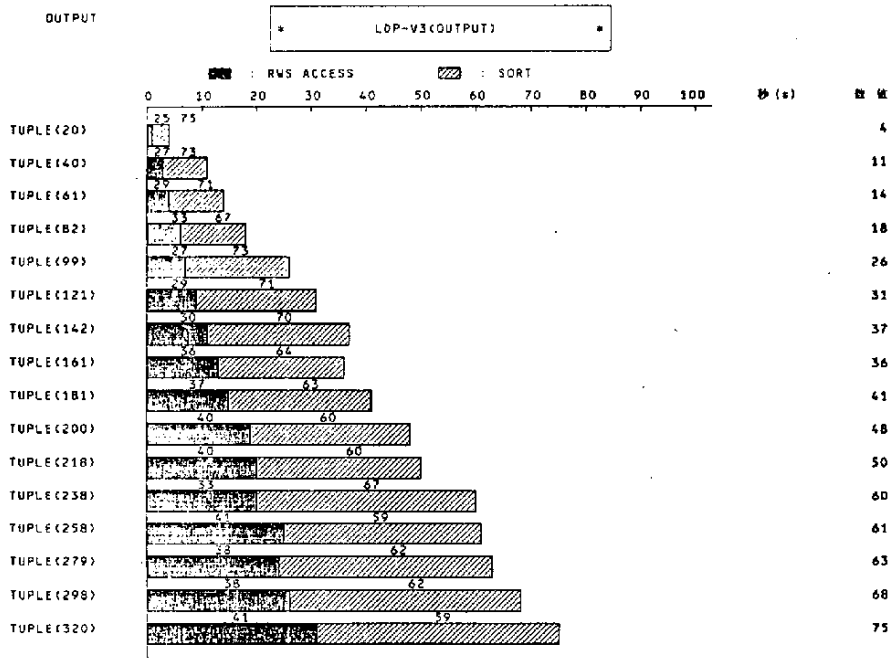


図5.17 OUTPUT処理の経過時間

G まとめ

LDP-V3として追加された、集積関数の処理、結果の関係作業領域(RWS)への格納と結果の編集出力処理の性能について評価を行った。

集積関数に対して、LDP-V3を構成するモジュールのうち隘路となるモジュールは、DMLGとDQGDであった。DMLGは、LDP-V1.5に対する改良に加えて、DMLパターンのコピーライブラリ化と繰り返しパターンの導入を行った。その結果、LDP-V1.5の改良版に比べて経過時間を3~4割短縮できた。

又、LDP-V3から生成されたCOBOL DMLプログラムの実行では、中間結果数が多い場合、結合処理の経過時間が全体の80~90%を占めてしまうことが明らかになった。これは、中間結果の1つに対して、他の中間結果の全てと結合処理を行っている為である。この為、中間結果の結合処理の効率化を検討している。

中間結果の結合処理の効率化では、結合条件となるgroup-by属性で中間結果をソートし、結合処理を行う回数を減らすことを考えている。同一の結合条件からなる中間結果に対しては、各中間結果を同時にreadし、結合条件を判定し、値が小さいものは読み飛ばす方法を考えている。1つでも中間結果がなくなれば、その時点で結合処理は終了する。但し、count、anyに対しては、条件を満たさないことも結果となるので、全ての組に対して結合処理を行う。

結果の関係作業領域(RWS)への格納では、格納件数が300件で、経過時間が約10分もかかっている。これは、格納領域の構造によるものと考えられる。経過時間を短縮する為には、組イメージのまま格納することを検討する。

結果の編集出力処理では、結果の分類処理がRWSからのアクセス処理の2倍程度経過時間を要している。出力コマンドから、結果が表示されるまでの間を短縮する為には、結果のファイルの出力と分類を無くした処理方法を検討する必要がある。その1つとして、RWSからのアクセス結果を、LDP内に蓄積し、メモリ内でソートを行う方法を考えている。

付記6 発表論文リスト

本プロジェクトにおいて発表した論文を以下に示す。

A. 昭和55年度

Takizawa, M., et al.,

“ Query Translation in Distributed Databases, ”
Proc. IFIP '80, 1980, PP. 451-456

滝沢

“ ローカルネットワークを用いた分散型データベースシステムにおける問合せ分割 ”
第22回IPSJ全国大会 1981, PP. 455-456

滝沢、鈴木

“ CODASYL DBTGデータベースシステムに対するリレーショナルモデル
インタフェース

— 設計とアクセス問題 — ”

ibid, 1981 PP. 435-436

滝沢

“ 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 — ”

情報の記憶と利用に関する理論的研究

(京大数理解析研講究録423)

京都大学数理解析研究所 1981 PP. 81-105

田中、滝沢

「 オフィス情報処理におけるフォームフローシステム 」

ibid. PP. 106-115

B. 昭和56年度

Takizawa, M.,

“ 分散型データベースシステムJDDBS-IIと通信処理 ”

電子通信学会 AL81-22, 1981

滝沢、増永、野口

“CODASYLデータベースシステムに対する非手続的インタフェースについて”

電子通信学会 AL81-67、 1981

横塚、滝沢

“CODASYLデータベースシステムに対するリレーショナルインタフェースシステム
LDP-V1.5の評価について”

第23回IPSJ全国大会、 1981年、 PP. 503~504

滝沢、田中

“CODASYLデータベースシステムに対する更新機能を備えたリレーショナル
インタフェースについて”

ibid. PP. 505~506

Takizawa, M.,

“Distributed Database-System JDDBS-I”

JIPDEC REPORT winter 1981

滝沢、横塚

“LDP-V1.5の総合評価”

データ・セマンティクスの理論と実際に関する研究

(数理解析研究所講究録461)

京都大学数理解析研究所 1982、PP. 127-150

滝沢

“分散型データベースシステムJDDBSの設計と実現について”

京都大学大型計算機センター 1982、PP. 48-67

第13回研究セミナー

横塚、滝沢

“LDP-V1.5のアクセスパス生成の最適化と総合評価”

第24回IPSJ全国大会 1982、PP. 473~474

C. 昭和57年度

Takizawa, M. ,

" Distribution Problems in Distributed Databases-Integration
and Query Decomposition "

JIP (IPSJ), VOL. 5、№3、1982. PP. 139-147

滝沢、横塚、鈴木

" CODASYL データベースに対する関係インタフェースシステム
(LDP-V1.5)の設計と実現 "

IPSJ 論文誌、VOL. 23、№3、1982. PP. 665-675

Takizawa, M,

" 放送通信機能を用いた分散型データベースシステムの通信処理実現について、"

IPSJ DBS 研資31、1982

Takizawa, M and Noguchi, S,

" CODASYLデータベースシステムに対する非手続的更新インタフェースの基本概念、"

Jipdec TR 10/82、1982.

滝沢、横塚

「CADASYL DBS上の非手続的更新インタフェース(LDP-V2)の設計と実現
について」

第25回 IPSJ全国大会 1982. PP. 615~616

滝沢、横塚

「JDDBSにおける通信処理の実現について」

ibid. PP. 617~618

Takizawa. M

" Heterogeneous Distributed Database System JDDBS "

Computer Science & Technology (Kitagawa. T. ed.)

Ohm-North-Holland. 1983

滝沢

“異種分散型データベースシステム JDDBS の全体アーキテクチャ”

第 26 回 IPSJ 全国大会 1983 PP. 659~660

滝沢、横塚、鈴木

“JDDBS における分散問合せ処理システム (GDP) の実現と評価”

ibid. PP. 661~662

滝沢、横塚、鈴木

“JDDBS における問合せ変換システム (LDP) の実現と評価”

ibid. PP. 663~664

滝沢、野口

“JDDBS における更新変換システム (LDP) の実現と評価”

ibid. PP. 665~666

APPENDIX 7

Distributed Database System - JDDBS

Abstract

Heterogeneity and distribution problems must be solved in order to realize a distributed database system (DDBS) consisting of various database systems (DBS's). This paper presents a DDBS (called JDDBS) architecture that is designed to integrate heterogeneous DBS's. Algorithms for the schema translation of a CODASYL schema into a common relational view as well as the translation of relational retrieval/update queries into COBOL DML programs are presented. Also, as the distribution problem, a definition of a global view over local views and distributed query processing based on broadcast communications, are discussed.

1. Introduction

A distributed database system (DDBS) is defined as a system consisting of more than one database system (DBS) with a semantically correlated database connected by a network, and one virtual DBS provided for users. The DBS is assumed to be a blackbox providing users with a data model, i.e., a data structure and operation language. Heterogeneity and distribution problems must be solved before the DDBS can be realized. The first problem is due to differences in the DBS's data models, and the second is posed by the distribution of multiple DBS's. The solution to these problems is a four-schema structure,^{16,17)} i.e., by defining common model views on individual DBS's, and by defining a global view over these local views, the first and second problems are solved, respectively.

SDD-1¹³⁾ and Polypheme¹⁾ have already been applied. They are composed of homogeneous relational DBS's connected by global packet-switched networks, like ARPANET. These applications demonstrate the the feasibility of the homogeneous DDBS.

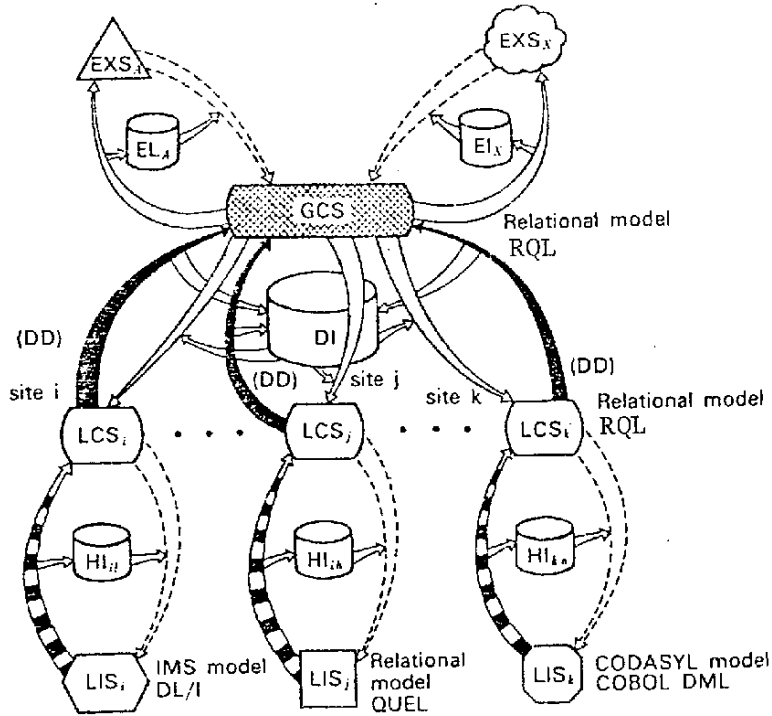
The proposed DDBS is called JDDBS (Jipdec DDBS).^{16,22)} It

has been under development since 1977 and has the purpose of integrating heterogeneous DBS's using various networks. A relational interface has already been implemented over the CODASYL DBS for solving the heterogeneous problem. This is called a local database processor (LDP). This processor provides users with full relational operations including aggregate and update operations on the relational view equivalent to the CODASYL data structure. The system⁴⁾ in Multibase¹⁴⁾ is a similar interface, but it can only process tree retrieval queries. Currently, the processor called a global database processor (GDP)^{19,20)} for distributed query processing is being employed. This is based on broadcast communications provided physically by local networks.

2. JDDBS Architecture

2.1 Schema-layers

The distribution and heterogeneity problems are solved by JDDBS, which consists of four schema-layers^{16,17)}: global conceptual, local internal, and external schemas as shown in Fig. 1. In this paper, a data model means a data structure and its operations, e.g., relational,³⁾ CODASYL,²⁾ and IMS models. A database system (DBS) is assumed to be a blackbox providing a data model for users. The local internal schema (LIS) corresponds to an existing DBS. Heterogeneity is defined as the differences in data structures and operations among DBS's. These differences are solved by defining common model views called local conceptual schemas (LCS's) of the existing DBS's. A relational model is adopted as the common model, because it provides simple and non-procedural means for describing and manipulating data, as well as a closure property. At the LCS level, every DBS can be virtualized into a relational DBS. The process of defining LCS's from the LIS's is called homogenization. The LCS description and as well as the LIS's and LCS's mapping information are called heterogeneity information (HI).



→: specialization

---→: homogenization

→: integration

---→: query translation (QT)

→: query decomposition

HI : heterogeneity information DD : distribution description

DI : distribution information GCS : global conceptual schema

LIS : local internal schema EXS : external schema

LCS : local conceptual schema

Fig. 1. Four-schema structure (FSS)

The distribution problem is solved by defining a global relational view called a global conceptual schema (GCS) throughout the LCS's in the DDBS. At the GCS level, a schema independent of distribution and heterogeneity, as well as an operation language are provided for users, i.e., the DDBS is virtualized into one logical DBS. This process is called integration. The GCS description and as well as GCS and LCS's mapping information are called the distribution information (DI).

The EXS is a description of specific data for an application from the GCS. The EXS description as well as mapping information

among the GCS and EXS are called external information (EI). This process is called a specialization. The EXS can be considered as an external schema of a single DBS,²³⁾ because the DDBS is already virtualized to a logical DBS at the GCS level.

We call a collection of HI's, DI, and EI's a DD/D of the DDBS. It is used in DDBS operations. Using the DI, a relational query on the GCS is decomposed into queries based on the LCS's which own the required data. These queries are executed by communications between DBS's via networks and by local DBS processing. This is called distributed query processing (DQP).^{5,8,19,20,24)} Each decomposed query is written in common relational language. Thus, decomposed queries are translated into operations that can be executed on the existing DBS using the HI and the results are returned as relations. This is called query translation (QT).^{17,18)}

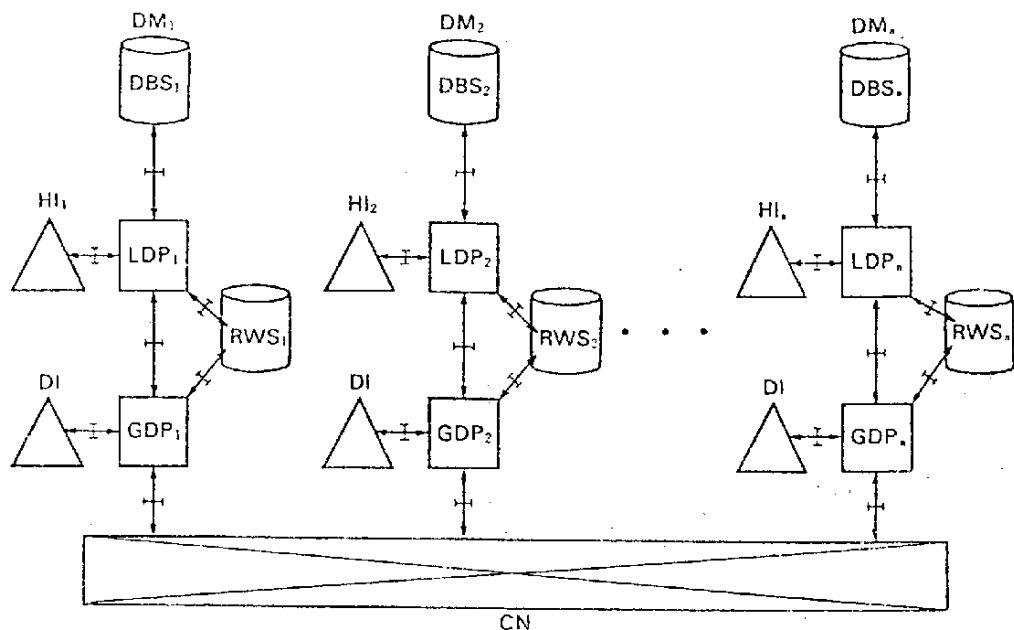
2.2 System architecture

The JDDBS consists of data modules $DM_1, \dots, DM_n (n \geq 1)$ connected by a communications network (CN) as shown in Fig. 2. The DM_i consists of six submodules, a database system (DBS_i), local database processor (LDP_i), global database processor (GDP_i), heterogeneity information (HI_i), distribution information (DI), and relational working storage (RWS_i). The DBS_i is an existing DBS that provides a data structure and operation language based on a data model. The LDP_i is a relational interface of the heterogeneous DBS_i , that translates LIS into LCS and generates HI_i , and which also translates a relational query into executable operations, executes them, and returns the results as relations to RWS_i . Since HI_i is specific to DBS_i , it is maintained locally in DM_i .

The GDP_i generates DI from a GCS definition by a GCS administrator, and decomposes a GCS query and executes it by communications with other GDP 's and local processing of relations stored in RWS_i . The DI is maintained in every DM in a fully-duplicated manner. The RWS_i is a file for storing relations. This functions not only as

an interface between LDP_i and GDP_i but also as a working file of GDP_i .

The CN provides high-reliable transparent communications between DM's in either a one-to-one or one-to-many (broadcast) mode.



HI : heterogeneity information	DM : data module
DI : distribution information	DBS : database system
CN : communications network	LDP : local database processor
	GDP : global database processor
	RWS : relational working storage

Fig. 2. JDDBS architecture

3. Local Database Processor (LDP)

A local database processor (LDP)^{17,18,21)} is a common relational interface system of the heterogeneous DBS. It provides users with facilities for the translation of LIS to LCS, and for the translation of relational queries into executable LIS operations. Since the CODASYL DBS is most prominent, it is taken up here.

3.1 LDP architecture

The LDP is composed of an HI processor (HIP) and query translation processor (QTP) as shown in Fig. 3. The HIP takes a CODASYL schema (LIS) and translates it into an equivalent relational schema (LCS). The HIP stores the LCS and the mapping information between both schemas as heterogeneity information (HI) in a relational form. The QTP takes a relational query based on the LCS, translates the query into a COBOL DML program using HI, executes it on the underlying CODASYL DBS, and returns the results as a relation to the RWS. The LDP^{18,21)} has been operational since 1981, and at present it provides full relational operations for users.

3.2 CODASYL model

The CODASYL schema can be represented as a directed graph whose nodes are record-types and arcs set-types. The record-type is denoted as $R(@R, t_1, \dots, t_m)$ ($m \geq 0$) where $@R$ is the db-key and $t_1, \dots, t_m$ the data-items. Let $dom(t)$ be a domain of item t . Possible tuples of the items values are called record (R)-occurrences. A set of them is called a record-occurrence (RO)-set,

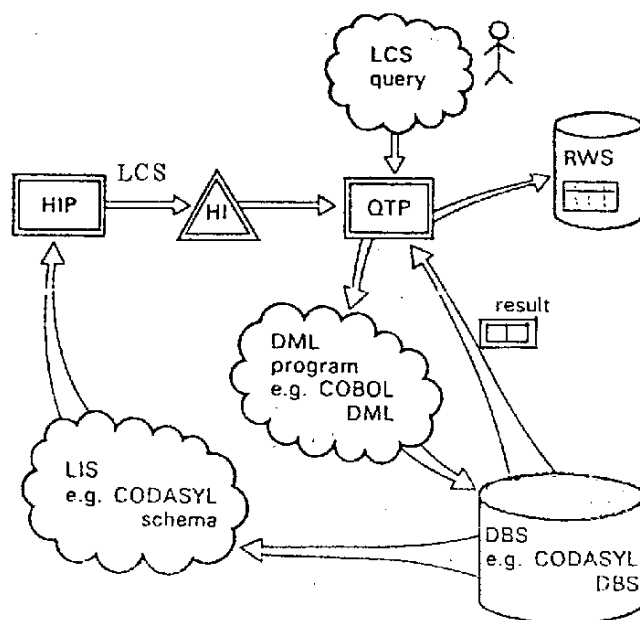


Fig. 3. Overview of the LDP

$R \subseteq \text{dom}(\text{@R}) \times \prod_{i=1}^m \text{dom}(t_i)$. For each R-occurrence r in R , let $t(r)$ be t 's value of r . Since R-occurrences in R can be identified by the db-key but not always by combination of the data-items, @R is a primary key. Since @R cannot be output to users, the record-type $\underline{R}(t_1, \dots, t_m)$ and $\text{@R}(r)$ dbk (r) are denoted for each r in R .

A partial function from a record-type $\underline{R}_2$ to $\underline{R}_1$ is represented as an arc $\underline{S}(\underline{R}_1, \underline{R}_2)$ called a set-type as shown in Fig. 4 (a). The $\underline{R}_1$ and $\underline{R}_2$ are called owner and member record-types of $\underline{S}$, respectively, and the extension of $\underline{S}$ is called a set-occurrence (SO)-set S . This is because every r_2 in $\underline{R}_2$ has at most one r_1 in $\underline{R}_1$ such that (r_1, r_2) in S . Each element in S is called a set(S)-occurrence. An n -ary relation among $n (\geq 2)$ record-types $\underline{R}_1, \dots, \underline{R}_n$ (not necessarily distinct) can be represented as n distinct set-types with a common member record-type $\underline{L}(t_1, \dots, t_m)$, $\underline{S}_i(\underline{R}_i, \underline{L}) (i=1, \dots, n)$ as shown in Fig. 4 (b). $\underline{L}$ may have data-items $t_1, \dots, t_m$. Since every R-occurrence I in $\underline{L}$ always has an owner r_i in $\underline{R}_i$ of $\underline{S}_i$ for $i=1, \dots, n$, $\underline{L}$ and $S_1, \dots, S_n$ represent an injection from $\underline{L}$ into $\underline{R}_1 \times \dots \times \underline{R}_n$. $\underline{L}$ is called a record-type for a link and $\underline{S}_1, \dots, \underline{S}_n$ set-types for link. A set $\{\underline{L}, S_1, \dots, S_n\}$ is a link-type and $\{\underline{L}, S_1, \dots, S_n\}$ a link-occurrence (LO)-set.

The CODASYL database is defined as a set of record-, set-, and link-occurrence sets as stated above.

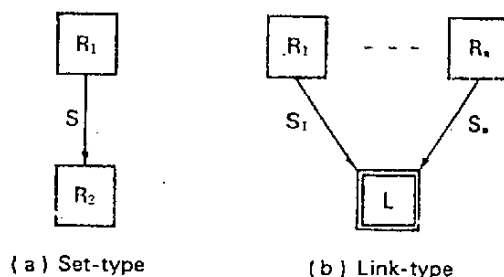


Fig. 4. CODASYL structure

Integrity constraints against updates are defined by membership-classes of set-types $\underline{S}(R_1, R_2)$, $mc(\underline{S})$. Suppose $mc(\underline{S})$ is either M(andatory)/A(utomatic) or O(ptional)/M(annual). If $mc(\underline{S}) = M/A$, a deletion of r_1 in R_1 is recursively propagated to deletions of members in R_2 of S . Also on storing a new occurrence r_2 in R_2 , an S-occurrence $(r_1 \in R_1, r_2)$ has to be stored in S . If $mc(\underline{S}) = O/M$, then R_1 , R_2 , and S can be updated independently. We assume that every set-type for link is typed M/A.

A CODASYL schema is shown in Fig. 5, single and double squares represent record-types and record-types for link, respectively. While solid and dotted arrows represent M/A and O/M set-types, respectively.

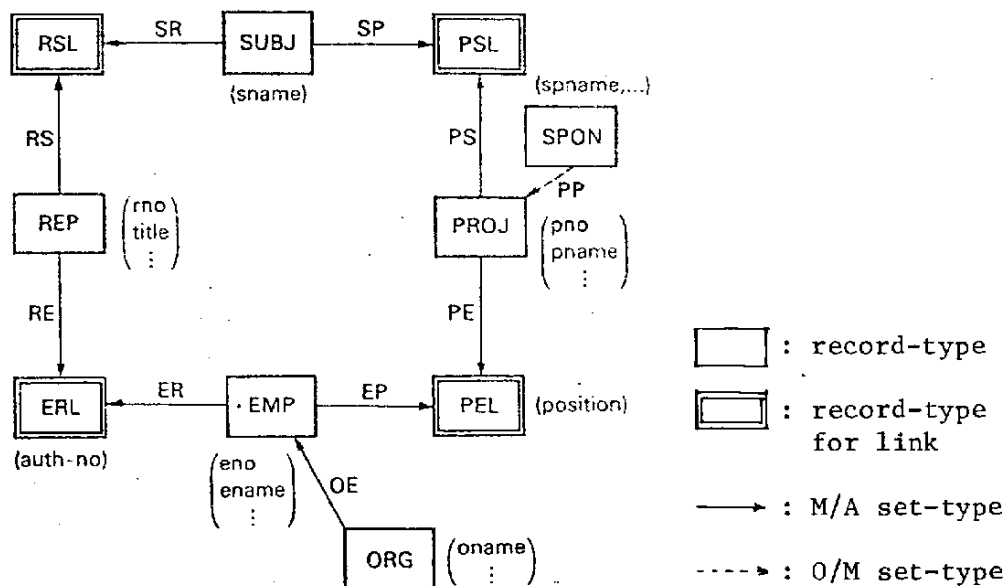


Fig. 5. CODASYL schema

3.3 Schema translation - homogenization

The HIP translates a CODASYL schema (LIS) into a relational schema (LCS) according to the following rules.

- (1) For every record-type $R(t_1, \dots, t_m)$, a relation scheme $R(@R_1, t_1, \dots, t_m)$ called an E-type is generated, where $@R$ is a primary attribute that takes a db-key and also

- a primary key.
- (2) For every set-type $\underline{S}(\underline{R}_1, \underline{R}_2)$ not for a link, a relation scheme $\underline{S}(@R_1, @R_2)$ is generated, where $@R_i$ is a primary key of the E-type corresponding to the record-type R_i ($i=1,2$). Relation scheme $\underline{S}$ is a T-type if $mc(\underline{S}) = M/A$, otherwise it is a P-type. Since S is a partial function from R_2 to R_1 , $@R_2$ is a primary key of the relation scheme S .
- (3) Concerning every link-type $\{L(t_1, \dots, t_m), \underline{S}_i(R_i, L), i=1, \dots, n\}$, a relation scheme $\underline{L}([S_1]@R_1, \dots, [S_n]@R_n, t_1, \dots, t_m)$, called a G-type is generated. If $@R_i = @R_j$ ($i \neq j$), S_k is prefixed to $@R_k$ ($k=i, j$). A set $\{@R_1, \dots, @R_n\}$ is a primary key of the relation scheme L .

E-type	<u>EMP</u>	(@E, eno, ename, ...)
	<u>SUBJ</u>	(@J, sname)
	<u>REP</u>	(@R, rno, title, ...)
	<u>ORG</u>	(@O, oname, ...)
	<u>PROJ</u>	(@P, pno, pname, ...)
	<u>SPON</u>	(@S, sname, ...)
T-type	<u>OE</u>	(@O, @E)
P-type	<u>PP</u>	(@S, @P)
G-type	<u>PEL</u>	(@P, @E, position)
	<u>ERL</u>	(@E, @R, auth-no)
	<u>RSL</u>	(@R, @J)
	<u>PSL</u>	(@P, @J)

Fig. 6. LCS

Here, the extensions of E-, T-, P-, and G-type are E-, T-, P-, and G- relations, respectively.

The corresponding information of the CODASYL schema and the LCS are stored in the HI. The LCS translated from Fig. 5 is shown in Fig. 6. From these rules, the following proposition holds.

Proposition 1: The data structure of the CODASYL schema and LCS generated from it (using the above rules) are in a one-to-one correspondence relation.

3.4 Query translation

The query translation processor (QTP) translates a relational LCS query into a COBOL DML program, executes it on the underlying CODASYL DBS, and returns the results as a relation to the RWS. The LCS queries and COBOL DML programs differ in data structure (relational vs. CODASYL) and access unit (set-at-a-time vs. record-at-a-time). In the approach to solve these differences,^{18,21)} a non-procedural query of the CODASYL data structure is introduced. By describing the LCS query in the CODASYL one, the data structure difference is eliminated. This is called structure transformation. From the CODASYL query, the access path is found so as to minimize access cost. As a result, the difference in access units is eliminated. This is called access path generation. Based on the access path, a COBOL DML program is generated and executed on the CODASYL DBS, and the resulting relation is returned to the RWS.

3.4.1 LCS query An LCS query is written in a relational query language (RQL) similar to QUEL⁷⁾ as follows.

```
var( $x_1, X_1$ )...( $x_m, X_m$ );  
get into  $R$  ( $t\bar{L}$ ) where qual;
```

Here, $X_1, \dots, X_m$ ($m \geq 1$) is the LCS relations not necessarily distinct. The var defines a tuple variable x_i for X_i ($i=1, \dots, m$). In the get, an attribute of X_i is denoted as $x_i.a$ which is an attribute-(a) variable. The qualification is a formula composed of two kinds of predicates: join $x_i.a_i \theta x_j.a_j$ and restriction $x_i.a_i \theta v$ (v is a constant), where $\theta \in \{<, \leq, =, >, \geq, \neq\}$. The $t\bar{L}$ is a target-list which is a list of the a-variables for output. R is the resulting relation. In RQL, only = and $\neq$ are allowed as comparison operators over the primary attributes, since they are identifiers (db-keys). Also, arithmetic operations on the primary attributes are not allowed, and only "count" and "any" in the aggregates⁷⁾ can be applied to them.

For simplicity, the following restrictions are made. (1) Qual

is a conjunction of predicates, (2) there are no aggregates, and (3) only equi-joins on the primary attributes. The current version, LDP-V3, does not have restrictions (2) and (3).

An LCS query based on the LCS in Fig. 6, "find reports, members, and their organizations such that members join the projects on DDBS and wrote the reports on DDBS as the first author," is written in RQL as follows:

```
var(e,EMP)(r,REP)(j,SUBJ)(p,PROJ)(o,ORG)(er,ERL)(rj,RSL);
var(pj,PSL)(pe,PEL)(oe,OE);
get into R(o.ename, e.ename, r.rno) where j.sname="DDBS" and
j.@J=rj.@J and rj.@R=r.@R and r.@R=er.@R and er.@E=e.@E and
e.@E=pe.@E and pe.@P=p.@P and p.@P=pj.@P and pj.@J=j.@J and
e.@E=oe.@E and oe.@O=o.@O and er.auth-no = 1; (1)
```

3.4.2 CODASYL query language (CQL) A non-procedural query on the CODASYL data structure is written in a CODASYL query language (CQL) as follows:

```
var( $r_1, R_1$ )...( $r_m, R_m$ );
get into R( $t\ell$ ) where qual;
```

Here, let $R_1, \dots, R_m$ be RO-sets (not necessarily distinct) ($m \geq 1$). The var defines an occurrence-(o-)variable r_i ranging over R_i ($i=1, \dots, m$). Data-item t of R_i is denoted $r_i.t$, called an item-(i-)variable. For an occurrence, o , in R_i which r_i takes, the value of $r_i.t$ is $t(o)$. In the get, R is a result file, $t\ell$ a target-list, and qual a qualification. The $t\ell$ is a list of i -variables for output. Qual is a conjunction of two kinds of predicates: set-predicate $S(r_i, r_j)$ and restriction $r_i \theta v$ ($\theta \in \{<, \leq, =, \geq, >, \neq\}$, v is a constant). Let $\underline{S}(R_i, R_j)$ be a set-type, and r_k an o -variable ($k=i, j$). The set-predicate $S(r_i, r_j)$ defines a set of pairs ($o_i \in R_i, o_j \in R_j$) in $\underline{S}$. The restriction is the same as the LCS query. The qual generates a subset O of a cartesian product $R_1 \times \dots \times R_m$ which satisfies the qual, i.e., $O = \{r \mid r \in R_1 \times \dots \times R_m \wedge \text{qual}(r)\}$. The $t\ell$ projects O to the result file R on the items in the $t\ell$ with no

duplication.

The CODASYL query graph (CQG) describes the CODASYL query, whose nodes indicate O-variables and arcs set-predicates. Node r may have a restriction and a list of target items denoted by symbols $\equiv$ and $\rightarrow$, respectively. The query describing the same meaning as (1) is written in CQL as (2) and its CQG in Fig. 7.

var(e, EMP)(r, REP)($j, SUBJ$)($p, PROJ$)(o, ORG)(er, ERL)(rj, RSL)(pj, PSL)(pe, PEL); get into $R(o, oname, e.ename, r, rno)$ where $j.sname = "DDBS"$ and $RS(r, rj)$ and $SR(j, rj)$ and $SP(j, pj)$ and $PS(p, pj)$ and $PE(p, pe)$ and $EP(e, pe)$ and $RE(r, er)$ and $ER(e, er)$ and $OE(o, e)$ and $er.auth-no = 1$; ----- (2)

3.4.3 Structure transformation From Proposition 1, the data structures of the LCS and LIS are in a one-to-one correspondence relation. Thus, the LCS query can be translated into a CODASYL query by the following ST rules. Here, let E, B, G be an E-, T- or P-, and G-relation, respectively, and also, R, S , and $\{L, S_1, \dots, S_n\}$ the RO-, SO-, and LO-sets corresponding to the relations E, B , and G , respectively.

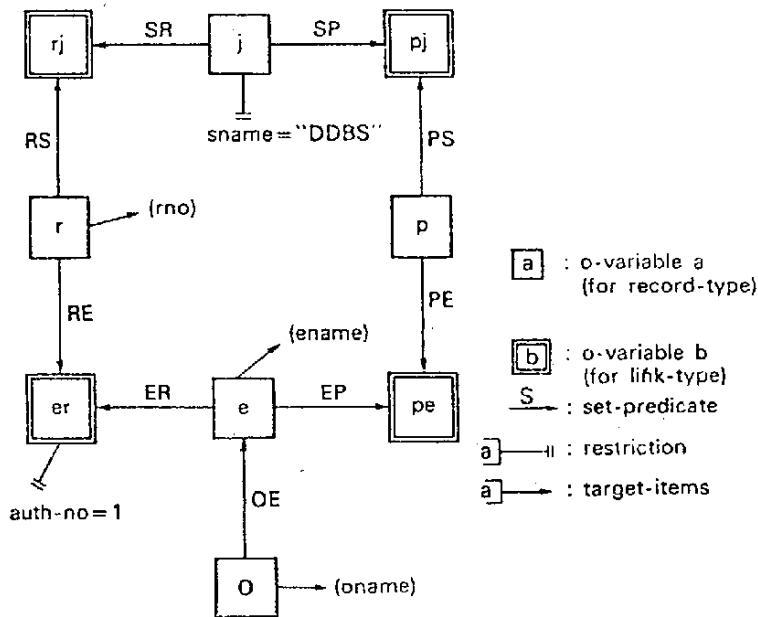


Fig. 7. CQG

ST rules:

- (1) Replace every tuple variable e of E by an o -variable r of the RO-set R .
- (2) For every variable b of $\underline{B}(@E_1, @E_2)$, where e_i is a variable of E_i ($i=1,2$), replace the conjunction $b.@E_1 = e_1.@E_1$ and $b.@E_2 = e_2.@E_2$ by a set predicate $S(r_1, r_2)$, where r_i is an o -variable of R_i ($i=1,2$).
- (3) Replace every variable g of $\underline{G}([S_1] @E_1, \dots, [S_n] @E_n, t_1, \dots, t_m)$, by an o -variable ℓ of the LO-set $L(t_1, \dots, t_m)$ and replace a predicate $e_i.@E_i = g.@E_i$ by a set-predicate $S_i(r_i, \ell)$ where r_i is an o -variable of R_i corresponding to E_i ($i=1, \dots, n$).

The LCS query (1) is transformed into the CODASYL query (2) by the ST rules. From proposition 1 and the ST rules, the following proposition holds.

Proposition 2: The LCS query and the CODASYL query generated by the ST rules have the same results, i.e., both have the same meaning.

3.4.4 Access tree (AT) and access path generation (APG) An access path is represented by an access tree (AT). The nodes in AT represent the o -variables. Node x may have restriction rf_x and target items list tl_x . The branches are directed and labelled by the set-predicates. Let $x_1, \dots, x_n$ be all the nodes in AT, and A_i an RO-set that x_i ranges over ($i=1, \dots, n$). The AT selects a subset O of a cartesian product $A_1 \times \dots \times A_n$, which satisfies all the restrictions and set-predicates. Then O is projected into result file R on the target items with no duplication.

The AT is generated from the CODASYL query graph (CQG) by uniquely searching the edges in the following depth-first manner.
ATG algorithm:

- (1) pushdown($\Lambda, \Lambda, \Lambda$); clear all marks of the CQG nodes and edges;

- (2) select a CQG node x s.t. $\text{cost}(\Lambda, x)$ as the minimum in all nodes; create an AT node t for x ;
- (3) if x is marked, mark x confluent else mark x used; $i \leftarrow 0$;
- (4) select an unmarked edge e among x and y s.t. $\text{cost}(e, x)$ as the minimum among all possible unmarked neighboring edges;
- (5) if found, then do mark e ; $i \leftarrow i+1$; pushdown (x, t, i) ; create an AT-node t' for y as the i -th child of t ; $t \leftarrow t'$; $x \leftarrow y$; go to 3 od;
- (6) else do popup (x, t, i) ; if $x \neq \Lambda$, then go to 4 else terminate od;

An AT generated from the CQG in Fig. 7 is shown in Fig. 8.

In the figure, the AT-nodes, j and j' , represent the same CQG-node j . Such AT-nodes as $x_1, \dots, x_n$ ($n \geq 2$) that represent the same CQG-node x of an RO-set X are called confluent nodes of x . The existence of $x_1, \dots, x_n$ implies the constraint that, for each tuple t in $X \times \dots \times X$, occurrences taken by $x_1, \dots, x_n$ in t must be the same. Therefore, AT represents the qualification as a conjunction of AT-node restrictions, set-predicates of the branches, and confluent node constraints. Such an AT has the following property.

Property 1: Given the confluent nodes $x_1, \dots, x_n$ of the CQG node x , every confluent node is included in the same subtree whose root node is one of the confluent nodes.

3.4.5 Access cost estimation Considering access cost estimation, it is assumed that elapse time of COBOL DML program execution depends on the number of accessed occurrences. In the system presented in this paper, AT is generated so as to decrease the number of accessed occurrences by heuristics using two kinds of statistics, which are connectivities of the set-types and selectivities of the AT-nodes. Let $b(p, c)$ be a branch labelled S whose parent, p , and child, c , range over RO-sets P and C , respectively. Also, $\text{card}(R)$ is a cardinality of RO-set R . Then, let σt_s

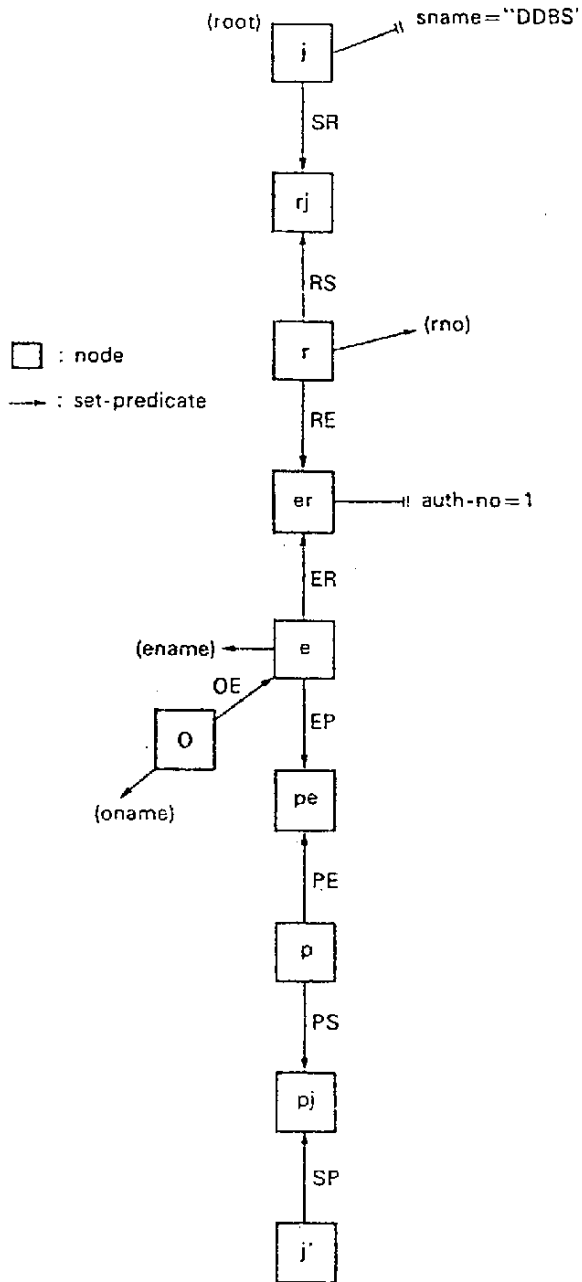


Fig. 8. Access tree of Fig. 7.

be the connectivity from p to c via S which is defined as follows:

$$ct_s = \begin{cases} pct_s & \text{if } P \text{ is an owner record-type of } S. \\ ict_s & \text{otherwise, i.e. } P \text{ is a member of } S. \end{cases}$$

$$pct_s = \frac{[\text{numb. of occurrences in } C \text{ which have an owner of } S]}{\text{card}(P)}$$

$$ict_s = \frac{[\text{numb. of occurrences in } P \text{ which have a child of } S]}{\text{card}(C)}$$

The pct_s gives the expected number of members of one p in P . The ict_s is the possibility that c in C has an owner of S . Let rf_a be a restriction formula of the AT-node a ranging over the RO-set A and st_a its selectivity. The st_a is defined as the possibility that an occurrence in A satisfies rf_a . The selectivity of $a.t = v$ is the ratio of $\text{card}(A[t])$ to $\text{card}(A)$. In Fig. 9, a_i, e_i, ct_i , and st_i are a CQG-node, edge between a_i and a_{i+1} , connectivity of e_i , and selectivity of the restriction of a_i , respectively. Suppose that a_1 was just selected by the ATG. In our algorithms the cost function in ATG is given as Eq. 3.

$$\text{cost}(e_1, a_2) = 1 + ct_1 * st * \min_{\forall ct_2} (1 + ct_2) \quad (3)$$

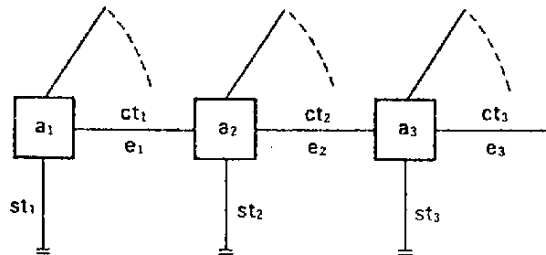


Fig. 9. Access path generation

3.4.6 DML generation (DMLG) A COBOL DML program is generated from the AT. Let b be a branch labelled S and c its child. The c is called a member node if it ranges over member RO-set C of S , otherwise, it is an owner node. For each AT-node c , the following notations are defined.

$$nptr(c) = \begin{cases} \text{the nearest member node-in the } c\text{'s ancestors if exist} \\ \text{a root node of the AT otherwise} \end{cases}$$

$$frtrn(c) = nptr(c)$$

$$srtrn(c) = \begin{cases} \text{a brother node just right to } c \text{ if exist} \\ nptr(c) \text{ otherwise} \end{cases}$$

For example, for AT-node o in Fig. 8, $nptr(o)=frtrn(o)=er$, $srtrn(o)=pe$, and for pe , $nptr(pe)=frtrn(pe)=srtrn(pe)=er$. The COBOL DML program can be generated by searching the AT in a preorder manner as follows.

DMLG algorithm:

- (1) $c \leftarrow$ a root of the AT; $b \leftarrow \Lambda$; pushdown (Λ, Λ) ;
- (2) $i \leftarrow 0$;
- (3) $i \leftarrow i+1$; find the i -th child d and its branch e of c ;
- (4) if found, then do DMLGT (c, b) ; pushdown (c, i) ; $c \leftarrow d$;
 $d \leftarrow e$; go to 2 od;
- (5) else do popup (c, i) ; if $a \neq \Lambda$, then go to 2 else terminate
od;

The procedure DMLGT (c, b) sequentially generates a DML block in a calling order from the DML patterns in Fig. 10; if c is a member-node, generate NEXT-block, otherwise OWNER-block. If $b=\Lambda$ (c is a root), CALC, sequential search via set-type or within area is preformed.

The CODASYL operations are based on the currencies. In our CODASYL query following the LCS query, different variables can be defined on the same RO-set. Furthermore, the confluent nodes may exist in the AT. Therefore, a DML block has to maintain the currency before passing control to the next block and recover it before accessing when its AT-node is a confluent member node or the RO-set of the o-variables. The currency can be maintained by the operation accept %20 from currency at the end of the DML-block. (%20 is a program variable declared with "usage is db-key.") Before executing find next, the currency can be restored by the operation

(1) NEXT-block

```
%01.  if %20 = T go to %02.
      move F to LFOUND. move T to %20. move F to %21.
      find first $01 within $02. go to %07.
%02.  if LFOUND = T, move T to %21.
%03.  find next $01 within $02.
%07.  if not end go to %04.
*      end
      move F to %20. move %21 to LFOUND.
      if %21 = T go to %07 else go to %06.
*      not end
%04.  get $01.
      if $03 and $04 next sentence else go to %03.
      perform RESULT (output).
*)    move T to LFOUND, %21. go to %03.
```

(2) OWNER-block

```
%01.  find owner within $02.
      if found go to %03.
*      not found
%02.  move F to LFOUND. go to %06.
*      found
%03.  get $01.
      if $03 and $04 next sentence else go to %02.
      perform RESULT.
*)    move T to LFOUND. go to %07.
```

where

%n = local label or variable in the DML block

\$n = parameter

\$00 = an area of a

\$01 = a record-type of a

\$02 = a set-type of a

\$03 = } restriction rf₂

\$04 = }

\$06 = %01 of frtrn(a)

\$07 = %01 of srtrn(b)

*) the last statement marked is output when the node
a is a leaf of the AT.

Fig. 10 COBOL DML patterns

find \$01; db-key is %20.

As stated in 4, occurrences by confluent nodes of the same CQG node have to be the same. Since Property 1 holds for AT, it is easy to verify this. Let $x_1, \dots, x_n$ be the confluent nodes of a CQG node x and x_1 a root of the subtree in which $x_2, \dots, x_n$ are included. In our DML, for an occurrence o_1 by x_1 , $k(\geq 1)$ occurrences $o_{i,j}, \dots, o_{i,k}$ of x_i are accessed ($i=1, \dots, n$). Hence, on accessing each $o_{i,j}$ ($j=1, \dots, k$), it is sufficient to check the identity of o_1 and $o_{i,j}$ ($@X(o_1) = @X(o_{i,j})$). This is a practical property, because there is no need of any intermediate file for computing occurrences in order to verify the identity of their occurrences.

3.4.7 COBOL DML program execution The LDP sends a job control language (JCL) to compile, link and execute the generated COBOL DML program to the host system either by TSS protocol of the in-house packed-switched network JIPNET or by an IPF (of FACOM M-170F). In the latter, the LDP running as a TSS user program can issue submit commands. The duplicates of the results are sorted out by the COBOL program. The resulting relation is stored in the RWS which is also maintained by CODASYL DBS.

3.5 Update translation

The current version LDP-V3 also can translate the non-procedural RQL updates on the LCS into COBOL DML programs.

3.5.1 CODASYL operations Let $\underline{R}$ be a record-type and R its RO-set. For every r in R , let $ms(r, R)$ be a union of a set of set-occurrences including r , and a set of r 's members r' in R' of M/A set-type $\underline{S}(\underline{R}, \underline{R}')$ and $ms(r', R')$. For every $\underline{R}$, let $Ts(\underline{R})$ be a set of pairs of M/A set-type $\underline{S}(\underline{R}', \underline{R})$ and its $\underline{R}'$. Here, let $\underline{R}(t_1, \dots, t_m)$, $\underline{S}(\underline{R}_1, \underline{R}_2)$, and $\{\underline{L}(t_1, \dots, t_m), \underline{S}_i(\underline{R}_i, \underline{L}) \ (i=1, \dots, n)\}$ be a record-type, set-type, and link-type, respectively. The logical operations are defined as follows.

(1) $del(r, R)$; $ms(r, R)$ are deleted.

(2) $del(s, S)$; $s=(r_1, r_2) \in S$ is deleted. If $mc(\underline{S})=M/A$,

- $r_2 \in R_2$ and $ms(r_2, R_2)$ are also deleted.
- (3) $del(l, L)$; $l \in L$ and $(r_i, l) \in S_i (i=1, \dots, n)$ are deleted.
 - (4) $app(r, R | r_1 \in R_1, \dots, r_n \in R_n)$; If $\exists r_i \in R_i$ for $\forall (S_i, R_i) \in O_S(\underline{R})$, r is not only stored in R , but also (r_i, r) is stored in $S_i (i=1, \dots, n)$.
 - (5) $app(s, S | r_1 \in R_1, r_2 \in R_2)$; If $mc(\underline{S}) = 0/M$ and $r_i \in R_i (i=1, 2)$, $s = (r_1, r_2)$ is stored in S_i .
 - (6) $app(l, L | r_1 \in R_1, \dots, r_n \in R_n)$; the same as (4).
 - (7) $rep(r, r', R)$; The data-item values of $r \in R$ are replaced by r' .
 - (8) $rep(s, s', S)$; The owner of s is replaced by s' .
 - (9) $rep(l, l', L)$; The data-item values of $l \in L$ are replaced by l' .

The logical operations and their COBOL DML's are shown in Table 1.

3.5.2 LCS operations Let $\underline{R}(@\underline{R}, t_1, \dots, t_m)$, $\underline{S}(@\underline{R}_1, @\underline{R}_2)$, and $\underline{L}(@\underline{R}_1, \dots, @\underline{R}_n, t_1, \dots, t_m)$ be an E-, T- or P-, and G-type relation schemes, respectively. For every $r \in R$, let $ts(r, R)$ be a set of T-, P-, or G-type tuples t such that $@R(t) = @R(r)$, and E-type tuples $r' \in R'$ and $ts(r', R')$ such that some T-type $\underline{S}(@\underline{R}, @\underline{R}')$ exist where r and r' are related by $s \in S$. For every R , let $Ts(\underline{R})$ be a set of pairs of a T-type $\underline{S}(@\underline{R}', @\underline{R})$ and its $\underline{R}'$. The update operations are defined as follows.

- (1) $del(r, R)$; $ts(r, R)$ are deleted.
- (2) $del(s, S)$; $s \in S$ is deleted. If $S = T$ -type, $r_2 \in R_2$ such that $@R_2(r_2) = @R_2(s)$ and $ts(r_2, R_2)$ are also deleted.
- (3) $del(l, L)$; $l \in L$ is deleted.
- (4) $app(r, R | r_1 \in R_1, \dots, r_n \in R_n)$; If $\exists r_i \in R_i$ for $\forall (\underline{S}_i, \underline{R}_i) \in Ts(\underline{R})$, r is added to R and also to s_i , which relates r and r_i into $S_i (i=1, \dots, n)$.
- (5) $app(s, S | r_1 \in R_1, r_2 \in R_2)$; If $r_i \in R_i (i=1, 2)$, s which relates r and r_i is added to S .

- (6) $\text{app}(l, L | r_1 \in R_1, \dots, r_n \in R_n)$; If $r_i \in R_i$, l is stored in $L(i=1, \dots, n)$ so that $@R_i(l) = @R_i(r_i)$.
- (7) $\text{rep}(r, r', R)$; Non-key values of r are replaced by r' .
- (8) $\text{rep}(s, s', S)$; Non-key values of s are replaced by s' .
- (9) $\text{rep}(l, l', L)$; Non-key values of l are replaced by l' .

The RQL update operations are written as follows.

- (1) delete x where qual ; Here, x is a tuple variable of an LCS relation X . For t, X which satisfies qual , $\text{del}(t, X)$;
- (2) append to $R(t_1 = \text{exp}_1, \dots, t_m = \text{exp}_m)$ where qual ; Here, $R(@R, t_1, \dots, t_m)$ and $TS(R) = \{ (S_i, R_i) | i=1, \dots, n \}$. For $\forall (r_1 \in R_1, \dots, r_n \in R_n, t_1, \dots, t_m)$ which satisfy qual , $\text{app}((t_1, \dots, t_m) | r_1 \in R_1, \dots, r_n \in R_n)$;
- (3) append to $S(@R_1 = r_1.@R_1, @R_2 = r_2.@R_2)$ where qual ; Here, S is P-type and r_i is a tuple variable of an E-type $R_i (i=1, 2)$. For $\forall (r_1 \in R_1, r_2 \in R_2)$ which satisfy qual , $\text{app}(s, S | r_1 \in R_1, r_2 \in R_2)$;

Theorem: Every RQL operation including retrieval and update on the LCS data structure can be translated into a COBOL DML program on the equivalent CODASYL data structure.

3.6 Evaluation

The LDP is programmed in PL/I with source size of about 10,000 and object size of about 250 KB. It is operational on AIM (FACOM M-170F) and ADBS (NEC ACOS-700) of a CODASYL type. The evaluation is designed to calrify the performance overhead and user gain when accessing the CODASYL DBS through the LDP. The elapse time for generating COBOL programs from LCS queries vs. the number of tuple variables referenced is shown in Table 1. This shows that the overhead is less than one minute. Furthermore, the LDP consumes 30 ~ 40% of the elapse time for structure transformation, 60 ~ 70% for COBOL DML generation, and less than 1% for access path generation. That is the writing of DML block in the file and the non-numeric processing for manipulating trees and graphs dominate LDP

performance.

Next, the life time is measured by two methods, i.e., one using the LDP and the other using the COBOL DMI. To answer identical questions in Japanese, it takes the LDP user an order of minutes, while it takes the COBOL user an order of days as shown in Table 1.

Table 1. RQL and COBOL DML (M-170F)

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=RQL, (h)=hour, (m)=minute, (s)=second

Therefore, it is concluded that the LDP overhead can be offset by increased software productivity. Still there is a problem that COBOL experts make better COBOL programs from the standpoint of step number and performance. One reason for the latter is that the statistics for cost estimation are not accurate. At present, the authors are trying to apply the statistics to actual situations by dynamically tuning them.

4. Global Database Processor (GDP)

A global database processor (GDP)^{19,20} provides facilities for generating distribution information (DI) from a GCS definition and for distributed query processing (DQP) using communications and local processings. A global conceptual schema (GCS) is defined in terms of LCS relations. It is assumed that GCS relations are defined as joins, restrictions, and projections of LCS relations in RQL. The DI processor (DIP) generates the GCS description as well as the mapping information between the GCS and LCS's from the GCS definition in the DI. Every DM has a full copy of the DI.

4.1 Communication network model

It is possible to make a model of a communications network's total communication time and response time. Let $C_{i\{j1, \dots, jn\}}(x)$ and $R_{i\{j1, \dots, jn\}}(x)$ be the communication time and the response time for transmitting data of volume x from DM_i to $DM_{j1}, \dots, DM_{jn}$ ($n \geq 1$), respectively. They are given in a conventional one-to-one network as follows.

$$C_{i\{j1, \dots, jn\}}(x) = n*(a + b*x) \quad (4)$$

$$a + b*x \leq R_{i\{j1, \dots, jn\}}(x) \leq n*(a + b*x) \quad (5)$$

In the best case, data are transmitted in parallel through n communication links. In the worst case, only one link is available at a time.

Networks which share a communication media like Ethernet¹¹⁾ and radionets, provide one-to-many (broadcast) communications. Communication time and response time are given as follows.

$$C_{i\{j1, \dots, jn\}}(x) = a + b*x = R_{i\{j1, \dots, jn\}}(x) \quad (6)$$

That is, time only depends on volume x but is independent of the number of destinations. The broadcast network is defined as a network for which Eq. (6) holds.

4.2 Distributed query processing (DQP)

In distributed query processing (DQP), first, a user's query on the GCS (GCS query) is taken by the GDP, called result GDP (RGDP). Then, it is translated into a query on the LCS (global LCS query) using the DI. This is called an expression translation. Next, the global LCS query is decomposed into LCS queries, each of which references relations in the LCS. The LCS query is executed by DM. If the DBS is heterogeneous, its LDP translates the query into an executable one, executes it on the DBS, and returns the resulting relation to the relational working storage (RWS). This is called initial local query processing (ILQP).⁸⁾ It is considered that, in the heterogeneous DDBS, the ILQP plays a role of deriving data as relations from the heterogeneous data structures than reducing the relations for communications. The global query is composed of the inter-DM joins based on the resulting relations of the ILQP's. By cooperation of the GDP's throughout the network, the result is obtained and is outputted to the user at the RGDP.

4.2.1 Assumptions The expression translation can be easily performed by query modification.¹⁵⁾ Also, the query translation in the ILQP has already been discussed in Section 4.2.3. Thus, inter-DM processing will be described next. From here, DQP means inter-DM processing. DQP will be discussed from the standpoint of communication mode (one-to-one vs. one-to-many), the decision method (static vs. dynamic), and the controller (centralized vs. distributed). In this paper, the DQP algorithm is based on the one-to-many communications, dynamic decision, and distributed control.

This paper only considers a global retrieval query after the ILQP, which is aggregate free and includes only equi-joins. Because of the wide bandwidth of broadcast networks (≥ 10 Kbps), the local processing cost is substantial compared to the global network.⁸⁾ It is assumed that the cost of equi-join of two relations, R_1 and R_2 , is proportional to $|R_1| + |R_2|$, where $|R_i|$ is

the size of $R_i (i=1,2)$. Finally, only one user is assumed to be allowed in the DDBS.

A schedule is defined as a sequence of transmissions and local processings of relations. A schedule has to be generated from the global query so as to achieve three objectives, i.e., to minimize total communication time and response time⁸⁾ as well as to keep the DI as static and small as possible.²⁰⁾

4.2.2 Distributed query processing using broadcast network

Since the broadcast network is used, if a DM issues its status information, all of the other DM's can grasp it, i.e., each DM can always know the global status of the DDBS. Hence, every DM can make the same decision on the next transmission at each stage of DQP. This implies that the DQP can be controlled in a completely distributed and dynamic decision manner. Thus, the DQP is simple and effective. Compared with other works^{5,8,23)}, the functions of the DQP are based on the broadcasting of data and control information, the distributed control of executions and communications, and the dynamic decision of a schedule.

4.2.3 Simple query A simple query is a query composed of equi-joins on $n (\geq 2)$ relations $R_1, \dots, R_n$ on join attribute a , where each R_i exists in a different DM_i . It can be processed as follows.

BA algorithm:

- (1) Let Σ be a set of relations $\{R_1, \dots, R_n\}$ and Φ be $\{DM_1, \dots, DM_n\}$.
- (2) For $\forall DM \in \Phi$, select R in DM s.t. $|R[a]|$ is the minimum in Σ using the global status (GS).
 $R' \leftarrow R[a]; \Sigma \leftarrow \Sigma - \{R\}; \Phi \leftarrow \Phi - \{DM\};$
- (3) For $\forall DM_i \in \Phi$, $R_i \leftarrow R_i[a=a]R'$; broadcast ACK_i with $R_i[a]$;
- (4) For $\forall DM_i \in \Phi$, wait for all ACK_j ; on receipt of ACK_j , the size $|R_j[a]|$ is set to the GS of DM_j ;

- (5) If $|\Sigma| \geq 1$, then go to 2 else ($\Sigma = \{R_i\}$) DM_i broadcasts $R_i[a]$ for DM_j ; R_j ; $R_j[a=a] (R_i[a])$;

On the completion of the BA, every DM_i has the same $R_i[a]$.

Suppose that relations are selected in an order $R_1, \dots, R_n$ in the BA and each join attribute a in R_i has the selectivity s_i .⁸⁾

Let C_{BA} be the total communication cost and P_{BA} the total processing cost. Let S_i be $s_1 * s_2 * \dots * s_i$ ($i \geq 1$). They are given as follows.

$$C_{BA} = |R_1[a]| + S_1 * |R_2[a]| + S_{n-1} * |R_n[a]| \quad (7)$$

$$P_{BA} = P_{1BA} + P_{2BA} + \dots + P_{n-1BA} + RP_{BA} \quad (8)$$

$$P_{iBA} = (n-i) * |R_i[a]| + |R_{i+1}[a]| + \dots + |R_n[a]|$$

$$RP_{BA} = |R[a]| + S_1 * |R_2[a]| + \dots + S_{n-2} * |R_{n-1}[a]| + (n-1) * |R_n[a]|$$

Here, $|R_i[a]|$ is the size of $R_i[a]$ at the beginning of the BA.

P_{iBA} is the summation of the processing costs for joining $R_i[a]$ and R_j ($j=i-1, \dots, n$). RP_{BA} is the total cost for joining $R_n[a]$ and R_j ($j=1, \dots, n-1$) to get the final result.

By broadcasting the first $R_1[a]$, every DM can hold it. Instead of transmitting $R_i[a]$ after the join of R_1 and R_i , a bit-map BM_i of the result $R_i[a]$ on $R_1[a]$ can be transmitted to reduce communications cost.⁹⁾ Furthermore, if the bit-map is also piggy-backed on the ACK (acknowledgement) message by every DM_i , a query can be processed by one transmission of $R_1[a]$ by DM_1 as well as $(n-1)$ ACK's by $DM_2, \dots, DM_n$. Such an algorithm, called BBA, is shown as follows.²²⁾

BBA algorithm:

- (1) Let Σ be a set of relations $\{R_1, \dots, R_n\}$.
- (2) For $\forall DM_j$, select R_k s.t. $R_k[a]$ is the minimum in; broadcast $R' (=R[a])$; $\Sigma \leftarrow \Sigma - \{R\}$; create BM_k with all bits on;
- (3) For $\forall R_i \in \Sigma$, $R_i \leftarrow R_i[a=a]R'$; create a bit map BM_i of $R_i[a]$; broadcast ACK _{i} with BM_i , wait for all ACK's;

- (4) On receipt of ACK_j , $BM_i \leftarrow BM_i \wedge BM_j$;
- (5) If all ACK's are received, restrict R_i on BM_i ;

This BBA reduces communications and processing costs. A general query including more than one join attribute can be processed by iterating the BBA for every join attribute.

4.3 Commitment control

Broadcast communications and distributed control make commitment control of transaction processing simple and reduce the required communications cost. Let $x_1, \dots, x_i$ be data objects ($n \geq 1$) to be updated by the transaction T where each x_i exists in a different DM_i ($i=1, \dots, n$). The atomic writing to these objects is controlled by the following BSC algorithm.²²⁾

BSC algorithm:

- (1) The update data of $x_1, \dots, x_n$ are obtained in the resulting DM(RDM) of transaction T by DQP as stated above.
- (2) RDM broadcasts a precommit ($x_1, \dots, x_n$) with the update data.
- (3) On receipt of the precommit, each DM_i saves it in the secure storage (SS_i).
- (4) If DM_i succeeds in doing it, DM_i broadcasts a precommitted, and waits for all precommitted messages.
- (5) If all received, DM_i physically updates DBS_i from the SS_i .

The two-phase commitment protocol⁶⁾ requires $4*n$ messages. However, this BSC algorithm requires only $1 + n$ messages, i.e., one precommit and n precommitted messages.

5. Conclusion

This paper presented a JDDBS architecture based on a four-schema structure, schema and query/update translations on CODASYL DBS, and the distributed query processing (DQP) based on broadcast communications. The local database processor (LDP) which is already operational on the CODASYL DBS's, AIM of M-170F and ADBS of ACOS-700, provides schema translation of the CODASYL to the

relational model, as well as full relational retrieval and update operations. These include aggregates over the relational view of the CODASYL schema for users. The evaluation of the LDP clarifies that total LDP productivity is superior to COBOL DML usage. Also, by demonstrating less than one minute overhead in elapse time, COBOL DML programs of a few thousands steps can be generated from relational commands of several lines. There still is a problem in that optimization based on statistics is not effective compared with the access path by human experts. Efforts are being made to improve cost estimation by dynamically tuning the statistics.

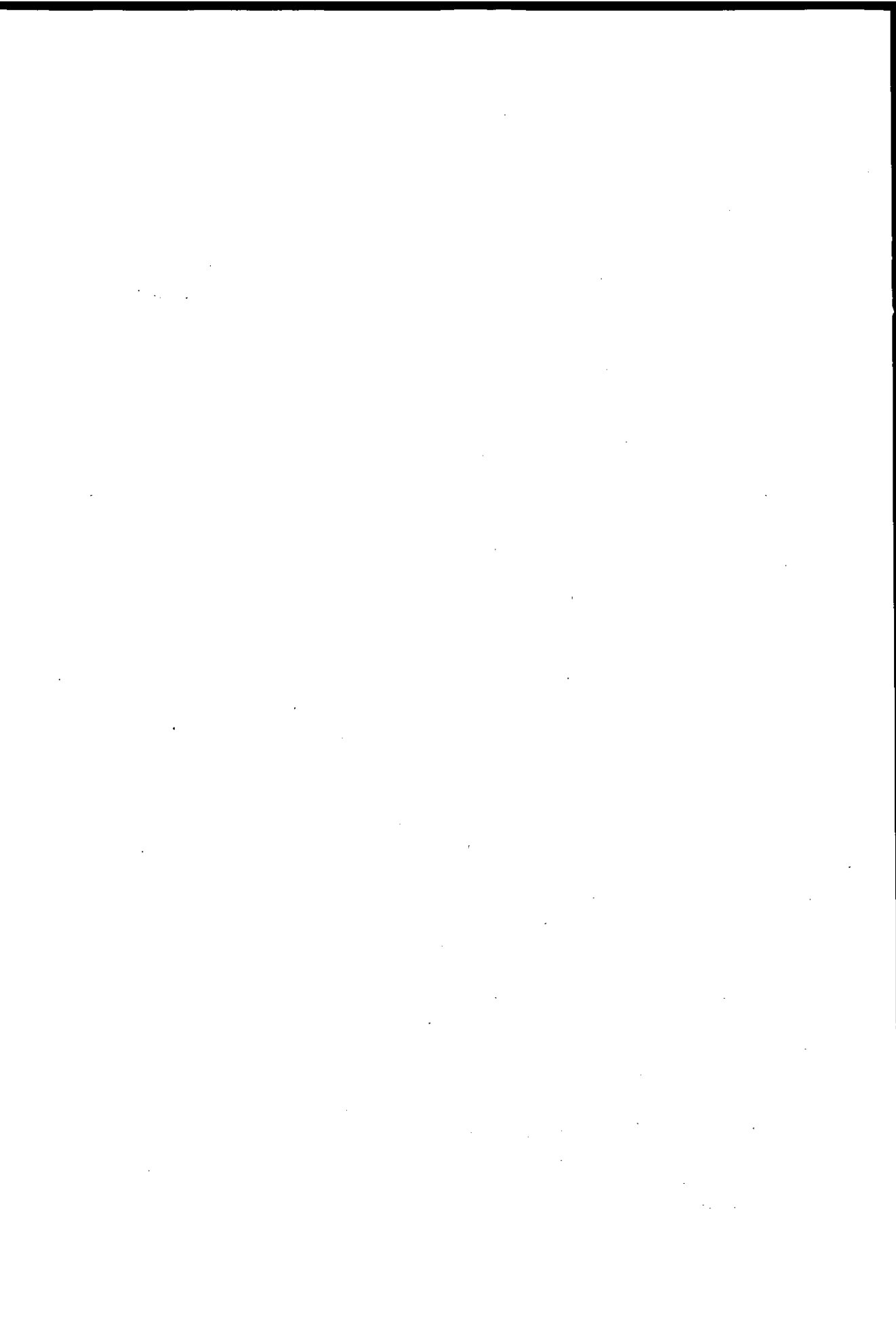
Different from conventional one-to-one networks, ones which share communications media, like Ethernet, provide broadcast communications. In this paper, the DQP algorithm based on the broadcast communications was presented. The commitment algorithm based on broadcasting, which decreases the number of messages, was also demonstrated. In the DDBS, it is concluded that broadcast communication makes management simple and effective.

This JDDBS is operational since February 1983. It is planned to be applied to the office information system.

References

- 1) Adiba, M. et al.: "An Overview of the Polypheme Distributed Database Management System", Proc. of IFIP (1980) 475.
- 2) "CODASYL Data Description Language of Development"(1973).
- 3) Codd, E.F.: "A Relational Model of Data for Large Shared Data Bank", CACM, 13 (1970) 337.
- 4) Dayal, U. and Goodman, N.: "Query Optimization for CODASYL Database System", ACM SIGMOD (1982) 138.
- 5) Goodman, N. et al.: "Query Processing in SDD-1: A System for Distributed Database", CCA-79-06, CCA (1979).
- 6) Gray, J.N.: Notes on Data Base Operating Systems, Operating Systems (Springer-Verlag)(1976) 393.
- 7) Held, G. et al.: "INGRES-A Relational Data Base System", AFIPS Conf. Proc. (1976) 409.

- 8) Hevner, A. and Yao, b.b.: "Query Processing on a Distributed Databases", Proc. of the 3rd Berkeley Workshop (1978) 91.
- 9) Kambayashi, Y. "Theory of Database", Computer Science & Technology, ed. by T. Kitagawa (Ohm & North-Holland, 1982).
- 10) Lebihan, J. et al.: "SIRIUS: A French Nationwide Project on Distributed Databases", Proc. of 6th VLDB (1980) 75.
- 11) Metcalfe, R. et al.: "Ethernet: Distributed Packet Switching for Local Computer Networks", CACM, 9 (1976) 395.
- 12) Olle, T.W.: "The CODASYL Approach to Data Base Management (John Wiley & Sons 1978).
- 13) Rothnie, J. et al.: "Introduction to a System for Distributed Databases: SDD-1", ACM TODS, 5 (1980) 1.
- 14) Smith, J. et al. "Multibase-Integrating Heterogeneous Distributed Database Systems", AFIPS Conf. Proc. (1981) 487.
- 15) Stonebraker, M. et al.: "The Design and Implementation of INGRES", ACM TODS, 1 (1978) 189.
- 16) Takizawa, M., Hamanaka, E. and Ito, T.: "Resource Integration and Data Sharing on Heterogeneous Resource Sharing System", Proc. of ICCS, 78 (1978) 253.
- 17) Takizawa, M. and Hamanaka, E.: "The Four-schema Structure Concept as the Gross Architecture of Distributed Databases and Heterogeneity Problems", JIP (IPSJ), 2 (1979) 134.
- 18) Takizawa, M. and Hamanaka, E.: "Query Translation in Distributed Databases", Proc. of IFIP, 80 (1980) 451.
- 19) Takizawa, M.: "Distributed Database System JDDBS-II and Communication Processing", TGAL81-22 (IECEJ) (1981) 43 [in Japanese].
- 20) Takizawa, M.: "Distribution Problems in Distributed Databases - Integration and Query Decomposition", J. Int. Systems (IPSJ) 5 (1982).
- 21) Takizawa, M., Yokotsuka, M., and Suzuki, M.: "On the Design and Implementation of a Relational Interface System over the CODASYL Database System", Trans. of IPS Japan, 23 (1982) [in Japanese].
- 22) Takizawa, M.: "Distributed Database System based on the Broadcast Communication", WGDBS (IPSJ) (1982) [in Japanese].
- 23) Tsichritzis, D. and Klug, A.: "The ANSI/X3/SPARC DBMS Framework, Information Systems , 3 (1978) 173.
- 24) Wong, E.: "Retrieving Dispersed Data from SDD-1: A System for Distributed Databases", Proc. of 2nd Berkeley Workshop (1977) 217.
- 25) Takizawa, M. and Noguchi, S.: "On the Design of the Non-procedural Interface System over the CODASYL Database System", TR82/10 (JIPDEC), (1982).



— 禁 無 断 転 載 —

昭和 58 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

57-S001

