計算機アーキテクチャ　90－1
（１９９１．１０．７）

高速ﾊｰﾄﾞｳｪｱｿｰﾀを用いた SQLｼｽﾃﾑの実現

伏見信也　岩崎浩文　安藤隆朗　佐藤重雄　小宮富士夫　樋口雅宏
三菱電機　ｺﾝﾋﾟｭｰﾀ製作所

ﾃﾞｰﾀﾍﾞｰｽﾌﾟﾛｾｯｻGREOを用いたSQLｼｽﾃﾑを試作した。本ｼｽﾃﾑは、GREOの特長であるﾊｰﾄﾞｳｪｱによる高速ｿｰﾄ機能を利用し、SQLによる問合せをｿｰﾄを中心とした実行形式に変換して高速に処理する。このために、SQLによる問合せからGREOのｿｰﾄや結合機能の組み合わせへのｺﾝﾊﾟｲﾙ方式、及びその際の最適化技法を開発した。本ｼｽﾃﾑは、ｵﾌｨｽｺﾝﾋﾟｭｰﾀMELCOM80ｼﾘｰｽﾞ上で試作され、現在評価中である。本稿では、本ｼｽﾃﾑの構成、GREO用のSQLｺﾝﾊﾟｲﾗの実現方式等について報告する。

Implementation of The SQL System Based on The High-Speed Sorter

Shinya Fushimi, Hirofumi Iwasaki, Takaaki Ando,
Shigeo Sato, Fujio Komiya, and Masahiro Higuchi

Computer Works, Mitsubishi Electric Corporation
325 Kamimachiya, Kamakura-Shi, Kanagawa 247, Japan

This paper reports the design and implementation of the new experimental SQL system. The system is unique in that it is based on the high-speed sorting function provided by the database processor called GREO. The new SQL compiler is developed to fully utilize the database primitives of GREO. The overall organization of the system is reported. Also, the details of the SQL compiler and the optimization techniques oriented to GREO are described.

1. まえがき

ﾃﾞｰﾀﾍﾞｰｽﾌﾟﾛｾｯｻGREO[1]を用いたSQLｼｽﾃﾑを試作した。本ｼｽﾃﾑは、GREOの特徴であるﾊｰﾄﾞｳｪｱｿｰﾀによる高速ｿｰﾄを中心に実現された検索専用SQLｼｽﾃﾑである。本稿ではそのｼｽﾃﾑ構成、SQLによる問合せのGREO命令群へのｺﾝﾊﾟｲﾙ技法等について報告する。

2. ﾘﾚｰｼｮﾅﾙﾃﾞｰﾀﾍﾞｰｽﾌﾟﾛｾｯｻGREO

2.1. ﾊｰﾄﾞｳｪｱ構成

GREOは東京大学生産技術研究所との並列ﾃﾞｰﾀﾍﾞｰｽ処理に関する共同研究の成果を製品化したものである。図1にGREOのﾊｰﾄﾞｳｪｱ構成を示す。GREOは当社のｵﾌｨｽｺﾝﾋﾟｭｰﾀMELCOM80ｼﾘｰｽﾞの付加ﾌﾟﾛｾｯｻとして使用される。GREOは、ﾊｰﾄﾞｳｪｱｿｰﾀ部とﾃﾞｰﾀﾍﾞｰｽ処理部により構成される。ﾊｰﾄﾞｳｪｱｿｰﾀ部は、LSIｿｰﾄﾌﾟﾛｾｯｻ[2]19石をﾊﾟｲﾌﾟﾗｲﾝ接続することにより構成され、8MB/512Kﾚｺｰﾄﾞまでのﾃﾞｰﾀを一度に高速ｿｰﾄすることができる。また、ﾃﾞｰﾀﾍﾞｰｽ処理部は、3つのﾏｲｸﾛﾌﾟﾛｾｯｻ(MC68020)、ﾛｰｶﾙ/共有ﾒﾓﾘ、ﾊｰﾄﾞｳｪｱｿｰﾀｲﾝﾀﾌｪｰｽ等により構成されたﾏﾙﾁﾌﾟﾛｾｯｻｱｰｷﾃｸﾁｬのﾃﾞｰﾀﾍﾞｰｽｴﾝｼﾞﾝである。これらﾏｲｸﾛﾌﾟﾛｾｯｻ群はｿｰﾄ処理の前後で選択、射影等の関係演算を実行し、また2ﾎﾟｰﾄ化されたｿｰﾀのﾒﾓﾘを用いることにより、ｿｰﾄ後のﾃﾞｰﾀのﾏｰｼﾞ、結合等の演算を高速に処理する。

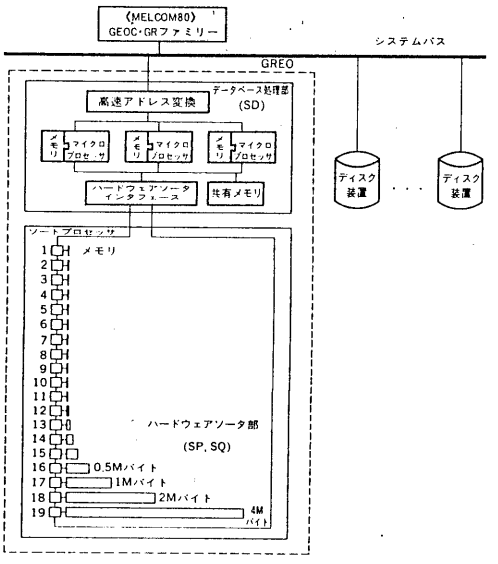

図1　GREOのﾊｰﾄﾞｳｪｱ構成

2.2. ｿﾌﾄｳｪｱ構成

GREOのｿﾌﾄｳｪｱ構成を図2に示す。GREOの3台のﾏｲｸﾛﾌﾟﾛｾｯｻ上には専用のﾏﾙﾁﾌﾟﾛｾｽ/ﾏﾙﾁﾌﾟﾛｾｯｻの制御OS(SY/OS)が搭載されている。GREOは固有のﾃﾞｰﾀﾍﾞｰｽ処理機能を持たず、ﾎｽﾄ計算機上に格納されているﾌﾟﾛｸﾞﾗﾑをﾀﾞｳﾝﾛｰﾄﾞし、これをGREO内部の3つのCPU上のOSにより多重ﾌﾟﾛｾｽとして動作させることによって所用の機能を実現する。このﾀﾞｳﾝﾛｰﾄﾞﾌﾟﾛｸﾞﾗﾑを拡張機能と呼び、対応するﾌﾟﾛｾｽをﾌｨﾙﾀと呼ぶ。このﾌｨﾙﾀに対してﾃﾞｨｽｸ上のﾃﾞｰﾀを連続的に入力することにより、ﾃﾞｰﾀの選択、結合等を実行する。現在、ｿｰﾄ(sorting)、ﾏｰｼﾞ(merger)、索引生成(index generation)、結合(join)等の機能を実現する拡張機能ﾌﾟﾛｸﾞﾗﾑが用意されている。以下ではｺﾝﾊﾟｲﾗでの慣例に従って、これらを各々ｿｰﾄ命令、結合命令等と呼ぶ。

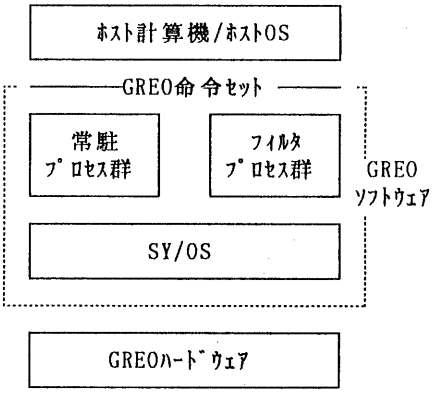

図2　GREOのｿﾌﾄｳｪｱ構成

2.3. 命令ｾｯﾄ

本試作では、GREOの命令の内、ｿｰﾄ命令、ﾏｰｼﾞ命令、結合命令のみを用いた。図3にGREOのｿｰﾄ命令、結合命令の構造を示す。

2.3.1. ｿｰﾄ命令

ｿｰﾄ命令は、入力されてくるﾃﾞｰﾀｽﾄﾘｰﾑを指定されたｷｰによりｿｰﾄして出力する。ｿｰﾄ命令では、ｿｰﾄの前処理としてﾃﾞｰﾀｽﾄﾘｰﾑに対する選択、射影、ｿｰﾄｷｰの抽出/生成機能、またｿｰﾄの後処理としてｿｰﾄ結果に対するｸﾞﾙｰﾌﾟ化、ｸﾞﾙｰﾌﾟ化により生成された各ｸﾞﾙｰﾌﾟに対する集合関数(aggregation)計算、重複値処理、

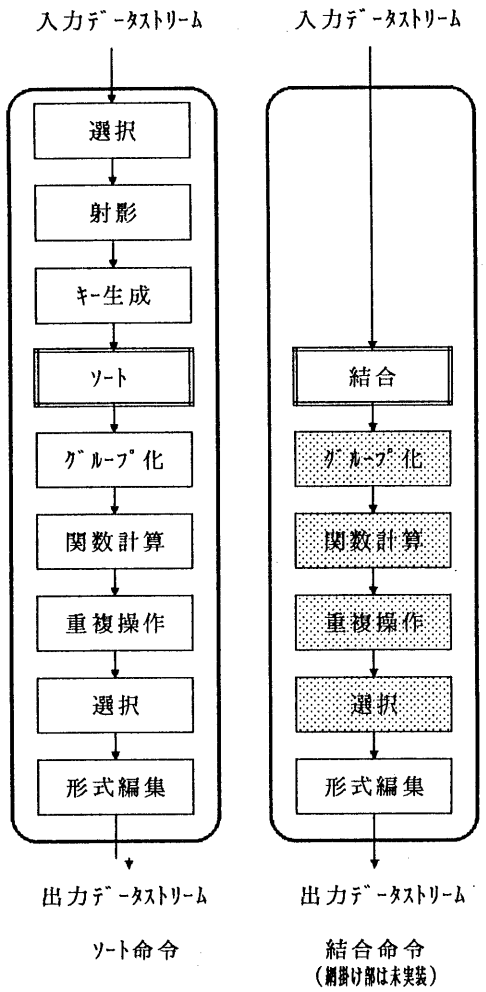

図3. GREOのｿｰﾄ/結合命令

ｸﾞﾙｰﾌﾟ化後の値に対する選択機能、及びこれらにより得られた個々の値を結果ﾚｺｰﾄﾞの形式に編集する機能が各々指定可能である。一度にｿｰﾄ可能なﾃﾞｰﾀ容量は現在8MBであるが、これを超えるﾃﾞｰﾀのｿｰﾄの際には8MB単位の連(string)を連続的に生成し、後にこれらをﾏｰｼﾞ命令により併合することによりｿｰﾄを完成させる。ﾏｰｼﾞ命令に対しては、ｿｰﾄ命令と同様な機能が指定可能であり、同様の議論が成立するため、ここでは詳細を省略する。

2.3.2. 結合命令

結合命令は、入力されてくる複数のｿｰﾄ済みのﾃﾞｰﾀｽﾄﾘｰﾑを指定されたｷｰにより結合する。結合命令は、ｿｰﾄ命令と同様な後処理が指定可能となるよう設計されているが、集合関数計算等、これら後処理の機能の一部は結合ｷｰとｸﾞﾙｰﾌﾟ化処理のｷｰが同一の場合のみ有効なものであり、使用頻度が低いとの判断から本試作では使用されていない。

3. ｼｽﾃﾑの概要

図4にｼｽﾃﾑの構成を示す。本ｼｽﾃﾑは、ｵﾌｨｽｺﾝﾋﾟｭｰﾀMELCOM80上に実現されている。以下、その主要構成要素について概略を述べる。

3.1. SQLのｵﾌﾞｼﾞｪｸﾄｺｰﾄﾞ

SQLのｺﾝﾊﾟｲﾙ結果を記述するｵﾌﾞｼﾞｪｸﾄｺｰﾄﾞをﾌﾟﾗﾝ(plan)と呼ぶ。ﾌﾟﾗﾝは、(一般に複数の)ｵﾍﾟﾚｰﾀ、親言語の変数値、ｵﾍﾟﾚｰﾀのﾌﾟﾗﾝ内での実行順序などの制御情報からなる。ﾌﾟﾗﾝは、ﾃｷｽﾄ部とﾃﾞｰﾀ部からなり、再入可能に構成されている。主としてｵﾍﾟﾚｰﾀ群とその実行順序に関する情報がﾌﾟﾗﾝﾃｷｽﾄ部におかれ、またｺﾝﾊﾟｲﾙ時には確定しない親言語の変数値記述や、検索した結果を表示するためのﾌｫｰﾏｯﾄ情報がﾌﾟﾗﾝのﾃﾞｰﾀ部に保持される。

ｵﾍﾟﾚｰﾀ(operator)は、GREOやSQL中核による処理の基本単位である。この内、ｿｰﾄ及び結合ｵﾍﾟﾚｰﾀは、主としてGREOに対するｿｰﾄ命令、結合命令を記述する。また、中間ﾌｧｲﾙの生成、削除、ﾚｺｰﾄﾞ単位のﾌｪｯﾁ等のｵﾍﾟﾚｰﾀが定義されている。ｵﾍﾟﾚｰﾀは、更新機能等、将来のSQLの完全仕様の実装を考慮して設計されている。

3.2. SQLｺﾝﾊﾟｲﾗ

SQLｺﾝﾊﾟｲﾗは、SQL文を解析し、これをﾌﾟﾗﾝに変換する。SQLｺﾝﾊﾟｲﾗは、与えられたSQL文から必要なｵﾍﾟﾚｰﾀ群を生成し、実行順序や同時実行可能な演算を解析して、結果をﾌﾟﾗﾝﾃｷｽﾄに格納する。SQL文と、それをｺﾝﾊﾟｲﾙして得られたﾌﾟﾗﾝの構造の例を図5に示す。ここでは、後述する選択述語の分配による最適化が仮定されている。これらｺﾝﾊﾟｲﾗの実現方式の詳細は4節で述べる。

3.3. SQL中核

SQL中核は、SQLｺﾝﾊﾟｲﾗによって生成されたﾌﾟﾗﾝを制御、実行する。SQL中核は、ﾌﾟﾗﾝの

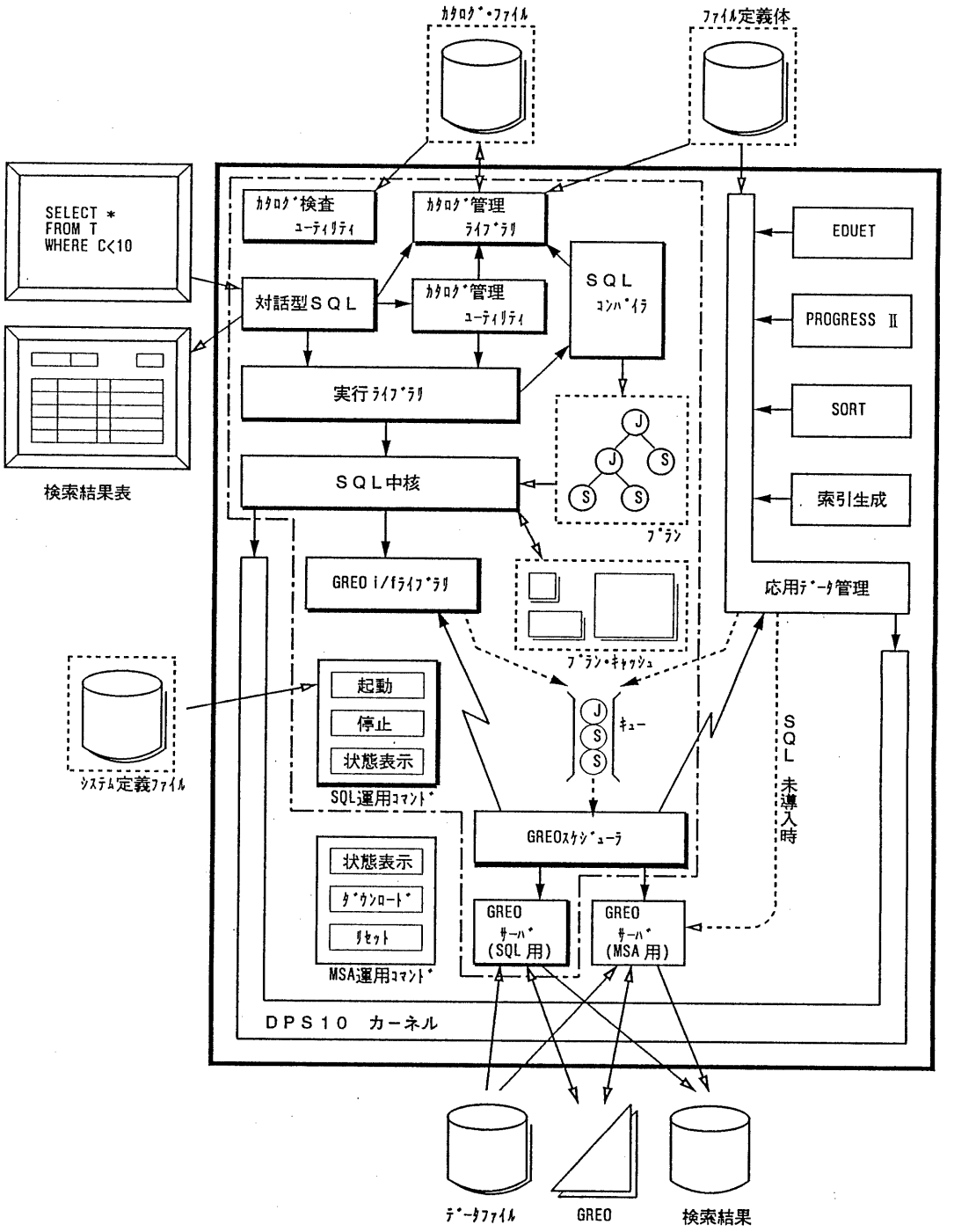

図4. SQLシステムの構成

○問合せ：

```
SELECT * FROM T, U
WHERE T.A = U.B AND U.B > 100
      AND T.A+U.A = 10
ORDER BY T.C                    ---------- ①
```

○ﾌﾟﾗﾝの構造：

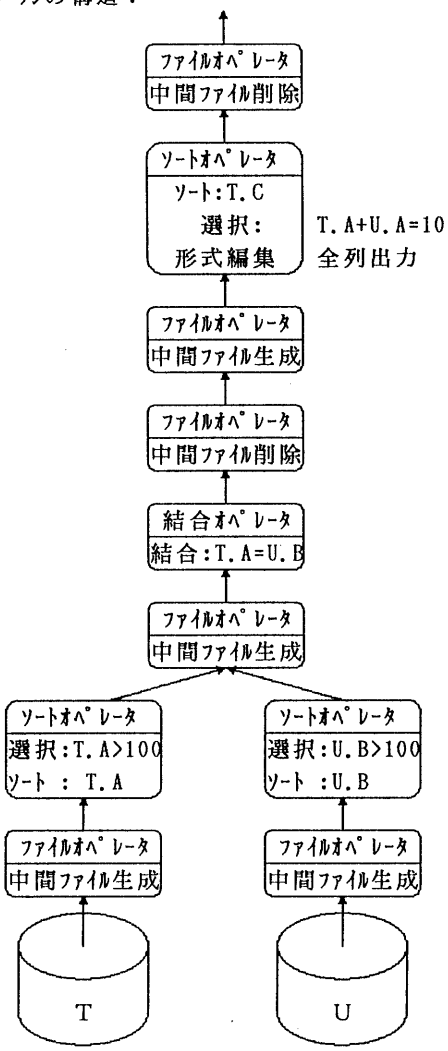

図5. SQL文とﾌﾟﾗﾝ

実行を開始する前に、まずそのﾃｷｽﾄ部をﾌﾟﾗﾝｷｬｯｼｭ(図4)にﾛｰﾄﾞする。さらに、ﾃﾞｰﾀ部をｱﾌﾟﾘｹｰｼｮﾝ毎に割付けられた領域に格納する。ﾃｷｽﾄ部を共通空間中のｷｬｯｼｭに配置することにより、複数のﾕｰｻﾞが同一のﾌﾟﾗﾝを実行する場合にﾃｷｽﾄを共有することが可能となる。定型業務等では、同一のSQL文が複数のｱﾌﾟﾘｹｰｼｮﾝから実行されることが多く、このような場合には一旦ﾛｰﾄﾞしたﾌﾟﾗﾝをｷｬｯｼｭに保持し、共有することによって実行時のｵｰﾊﾞﾍｯﾄﾞを軽減することが可能となる。

SQL中核はﾃﾞｰﾀ駆動の原理によりﾌﾟﾗﾝ中のｵﾍﾟﾚｰﾀ群の実行を制御する。即ち、まずﾌﾟﾗﾝ中の実行可能な葉(leaf)ｵﾍﾟﾚｰﾀを検索し、これに対してGREOｽｹｼﾞｭｰﾗに処理要求を発行する。その後、これら要求に対するGREOによる処理が終了し、対応するｵﾍﾟﾚｰﾀの親ｵﾍﾟﾚｰﾀが実行可能となると、親ｵﾍﾟﾚｰﾀの処理を起動する。最終的に全てのｵﾍﾟﾚｰﾀの実行がを完了すると、目的とする検索結果が得られる。

3.4. GREOｽｹｼﾞｭｰﾗとGREOｻｰﾊﾞ

GREOｽｹｼﾞｭｰﾗは、SQL中核からの要求に対しGREOの割付けを制御するGREOの資源管理ｽｹｼﾞｭｰﾗである。また、GREOｻｰﾊﾞは、GREOの提供する各命令毎にその実行を制御するﾃﾞｰﾀﾍﾞｰｽ機能単位のｻｰﾊﾞである。

GREOｽｹｼﾞｭｰﾗは、SQL中核より起動要求を受け取ると、使用可能なGREOを探し、要求されている機能に対応したGREOｻｰﾊﾞを起動する。複数の要求に対しては優先順位により実行順序を決定する。GREOｻｰﾊﾞは、検索対象ﾌｧｲﾙからGREOへのﾃﾞｰﾀ転送、GREOからの検索結果のﾌｧｲﾙへの転送等を制御する。処理が終了すると、これをGREOｽｹｼﾞｭｰﾗに通知する。GREOｽｹｼﾞｭｰﾗは当該処理の完了をSQL中核に対して伝えると共に、起動要求が待ち行列にあれば優先順位に従って対応するGREOｻｰﾊﾞを再び起動する。

本GREOｽｹｼﾞｭｰﾗにより、GREOが複数台搭載されているｼｽﾃﾑでは、複数のｵﾍﾟﾚｰﾀを同時に処理することが可能となる。例えば図5に示されたﾌﾟﾗﾝを単独で実行した場合には葉にあたる二つのｿｰﾄｵﾍﾟﾚｰﾀは同時に処理される。また、GREOの一部が故障した場合には、GREOｽｹｼﾞｭｰﾗより縮退運転が実現される。

4. SQLｺﾝﾊﾟｲﾗの実現方式

4.1. 概要

SQLｺﾝﾊﾟｲﾗは、構文解析、実行木の生成、最適化、ｺｰﾄﾞ生成の各部により構成されている。この内、構文解析等は通常のｺﾝﾊﾟｲﾗと同

様であり、ここでは記述を省略する。

本ｺﾝﾊﾟｲﾗには以下の特徴がある。

(1)全てのSQLによる検索要求を上記の2命令の組み合わせにｺﾝﾊﾟｲﾙする。特に、結合演算の実現ｱﾙｺﾞﾘｽﾞﾑとしては、ｿｰﾄ/ﾏｰｼﾞ結合のみを対象としている。

(2)可能な全ての等価変形に対し、問合せ処理途中で表同志の直積が必要となる表の集合を検出し、これが存在すればｴﾗｰとする。これは通常のｱﾌﾟﾘｹｰｼｮﾝでは直積を生成することは稀であり、また直積生成のために長時間GREOを使用することは資源管理の点で有利ではないことによる。

(3)等結合により結合される表の集合に対し、可能ならばその集合中の表に指定されている選択述語を等結合される全ての表に分配し、なるべく多数の表に対して予め入力ﾃﾞｰﾀの絞り込みを行う。

(4)複数の2-way結合を単一のn-way結合、nｷｰ結合に変換し、GREOへの一度のﾃﾞｰﾀｽﾄﾘｰﾑ転送により結合演算を実行可能とする。

尚、本ｼｽﾃﾑで対象としたSQL言語は、JISのSQL仕様[4]を基本とし、一部SQL2[5]の仕様を追加した。また、入れ子にわたる列の参照を含む副問合せに関しては、ﾌﾟﾘﾌﾟﾛｾｯｻによる標準化を仮定しており、ここでは議論しない。以下ではSQLの仕様に従い、表(table)、列(column)、行(row)等の用語を用いる。

4.2. 実行木の生成

実行木(Execution Strategy Tree)とは、問合せが参照している表の集合を結合演算により結合し、一つの出力表とするための結合順序を決定すると共に、これら各結合演算に付随して実行される選択演算等を確定するための木構造である。木の各ﾉｰﾄﾞは、演算結果としての表を表し、またその表を生成する結合述語、結合結果に対する選択述語を持つ。ここではﾉｰﾄﾞが表現する表、または結合結果の表の集合をﾉｰﾄﾞにﾗﾍﾞﾙとして付加し、ﾉｰﾄﾞの結合述語、選択述語を各々「J:」、「S:」により表す。本ｺﾝﾊﾟｲﾗにおける実行木の生成は、通常のｺﾝﾊﾟｲﾗの意味解析に対応する。

実行木生成の基本ｱﾙｺﾞﾘｽﾞﾑは、良く知られている関係代数木生成ｱﾙｺﾞﾘｽﾞﾑ[6]を拡張したものである。まず、WHERE句が比較述語等の基本述語のAND結合のみから構成される場合を考える。図5のSQL文①を例に説明する。

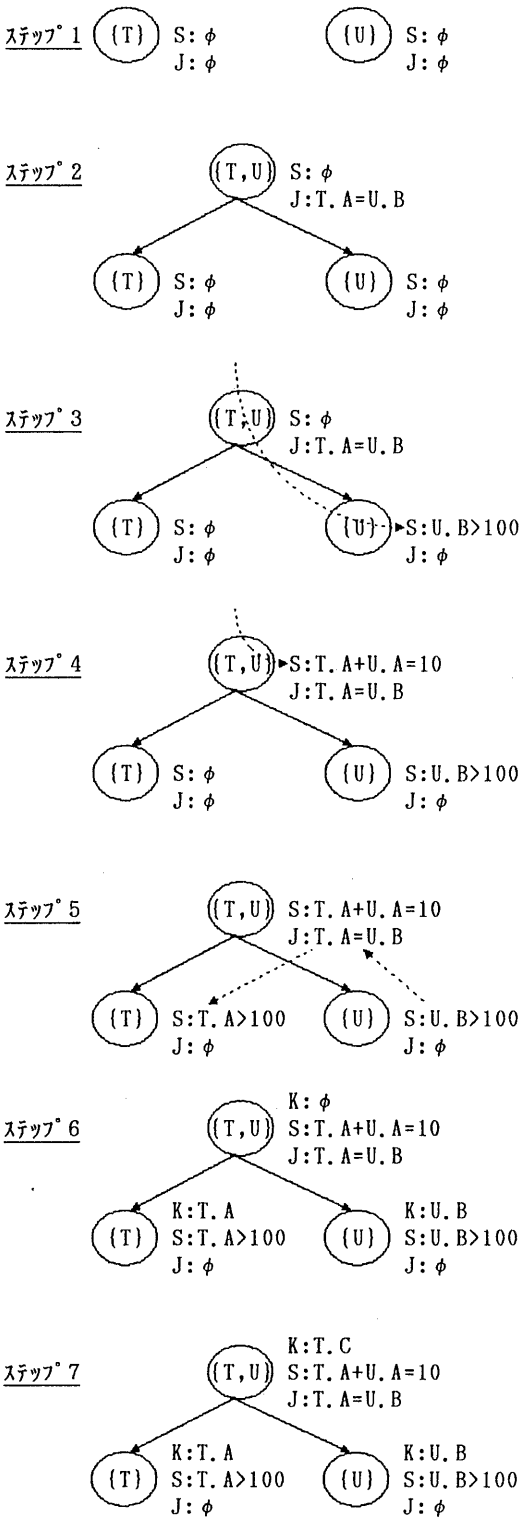

図6. SQL文のｺﾝﾊﾟｲﾙ

最初にFROM句内の各表に対応してﾉｰﾄﾞを生成し、これを初期化する。この例では図6ｽﾃｯﾌﾟ1に示すﾉｰﾄﾞの森(forest)が得られる。

次に、WHERE句中の各述語を次々と取り出し、述語の各々の辺が参照する表(の集合)をこの森から探す。この際、ANDで接続された基本述語の集合は実行順序に関して可換であり、適用順序に依らず同一の結果が生成されることが基本となる。この場合、T.A=U.Bを例にとると、この述語の左辺、右辺が参照する表の集合は各々{T}、{U}であり、各々の表集合を含むﾉｰﾄﾞが森に存在する。これはT.A=U.Bがこれらﾉｰﾄﾞが表現する表を結合することを意味する。このような述語が存在すれば新たにﾉｰﾄﾞを作る。このﾉｰﾄﾞは結合演算の結果を表し、結合されるﾉｰﾄﾞを新しいﾉｰﾄﾞの子に設定し、そのJ:ﾌｨｰﾙﾄﾞに当該述語を設定する。この例ではTとUの結合結果を表現する新しいﾉｰﾄﾞが生成され、TとUに対応するﾉｰﾄﾞが子に設定される。また、新しいﾉｰﾄﾞのJ:に述語T.A=U.Bが付加される。このﾉｰﾄﾞは、TとUの全ての列を含むと仮定される。これにより実行木は図6ｽﾃｯﾌﾟ2に示すものとなる。

次に、U.B>100を考える。この述語は{T,U}上の述語であるが、ﾉｰﾄﾞ{T,U}の右の子{U}の列のみを参照している。これから実行木上でこの述語を{T,U}から{U}に移動することができる。{U}は葉であり、これ以上述語は移動できず、この述語はﾉｰﾄﾞ{U}の選択述語としてS:に設定される。これは、選択述語を先に実行し、結合演算実行以前にﾃﾞｰﾀの絞り込みを行うことを意味する(図6ｽﾃｯﾌﾟ3)。

最後に、T.A+U.A=10は{T,U}上の述語であるが、その左辺は左右両方の子の列を参照しており、これ以上実行木上を移動できない。また{T,U}のJ:には既に結合述語が設定されているため、この述語は選択述語としてそのS:に付加される。これは、この述語がJ:に設定されている述語による結合結果に対する選択演算であることを意味する(図6ｽﾃｯﾌﾟ4)。

WHERE句にNOTを含む場合には、De Morganの法則によりこれを変形し、基本述語の論理を反転することで全体をNOTの無い形に変形する。WEHRE句にORを含む場合は、基本述語をORで接続した和項(disjunctive)をANDで接続した標準形(disjunctive normal form)に変形し、これら和項群に対して上記の基本ｱﾙｺﾞﾘｽﾞﾑを適用する。ここで和項により等結合がなされる場合があることに注意する。例えば、①のWHERE句が

「T.A+1 = U.A OR T.B*2 = U.B/U.C」

の場合、この述語全体により表TとUが結合される。これら和項による結合は、和項中の各述語による結合を個々に実行し、これらの結果をUNIONすることにより実現される。これから判るように、和項内の全ての基本述語の左辺、右辺が各々同一のﾉｰﾄﾞを参照するように和項内の基本述語の両辺を交換することができる場合に限って和項は結合述語となる。例えば、上の例のWHERE句が

「T.A+1 = U.A OR U.B/U.C = T.B*2」

である場合、ｺﾝﾊﾟｲﾗは、和項内の2番目の基本述語の左辺と右辺を交換することにより、和項内の全ての基本述語の左辺がTを、また全ての述語の右辺がUのみを参照していることを検出する。

更に、SQLでは、これらORによる結合に対して種々の表現が可能な点に注意する必要がある。例えば、①のWHERE句として

「T.A+1 IN (U.A+1, U.B-U.C)」

と

「T.A+1 = U.A+1 OR T.A+1 = U.B-U.C」

は等価であり、共にTとUを結合する結合述語となる。また、同様に

「T.A - U.A = 0」 は 「T.A = U.A」

と等価であり、結合述語と見なせるが、実用上の観点から本ｼｽﾃﾑでは結合述語判定の等価変形の対象としていない。

4.3. 最適化

本ｺﾝﾊﾟｲﾗは生成された実行木に対して以下の最適化を行う。

(1)実行木中の各ﾉｰﾄﾞに対し、その結合述語が和項でない場合、その任意の子のS:ﾌｨｰﾙﾄﾞに設定されている選択述語の内、元のﾉｰﾄﾞのJ:にある結合述語のいづれかの辺を対象としているものがあればこれを結合に関与

する全てのﾉｰﾄﾞに分配する。例えば、上の例①では結合の右辺はU.Bであり、またこれを対象とした述語U.B>100がその右の子に存在するから、これをその左の子に分配することができる(図6ｽﾃｯﾌﾟ5)。

(2)実行木中の各ﾉｰﾄﾞにおいて、J:とS:が共に設定されている場合には、これらの述語群を比較し、入れ換えを行って、最も結合による絞り込み効果が高いと判断される述語を結合述語としてJ:に再設定する。例えばJ:、S:が

J:T.A=U.A OR T.B = U.B
S:T.C=U.C

であれば、J:に設定されている結合述語にはUNIONが必要とされるため、S:にある述語の方が絞り込み効果が高いと判断でき、結合述語との入れ換えを行う。

(3)ﾉｰﾄﾞ群のJ:述語を単一のn-way結合に併合する。例えば、J0:T.A=U.A, J1:U.A=V.Aを J:T.A=U.A=V.Aに変換する。尚、この変形は本試作では実行木生成の前処理として実装されている。

(4)各ﾉｰﾄﾞに対し、J:が設定されている場合には、これをそのS:中の述語群と結合述語を併合し、単一のnｷｰ結合に変換することを試みる。例えば、

J:T.A=U.A　　J:(T.A,T.B)=(U.A,U.B)
S:T.B=U.B　⟶　S:φ

なる変形が可能である。

4.4. ｺｰﾄﾞ生成

以上により生成された実行木が、ｺｰﾄﾞ生成によりGREOのｿｰﾄ、結合命令に変換される。上述のように、これら命令には種々の補助機能が指定可能であり、ｺｰﾄﾞ生成ではこれら命令中の補助機能をなるべく有効に利用し、生成する命令の数の減少させることを試みる。例えば、実行木中のあるﾉｰﾄﾞの結合演算の実行のためには、結合ｷｰに対するｿｰﾄが必要であるが、このｿｰﾄの際に、ｿｰﾄ命令の補助機能を利用して、子ﾉｰﾄﾞの選択演算が実行可能である。このように、ｺｰﾄﾞ生成では、命令で指定可能な補助機能群に実行木に設定されている演算を最密充填することが目標となる。

このために、実行木のﾉｰﾄﾞに対して、ｿｰﾄｷｰﾌｨｰﾙﾄﾞ(K:)を追加し、各ﾉｰﾄﾞでは、結合→→選択→ｿｰﾄの順に指定された演算を実行するものとする。例えば上の例に対する実行木は、結合の前処理としてのｿｰﾄを考慮すると図6ｽﾃｯﾌﾟ6のように変形される。また、ORDER-BYやGROPU-BY、HAVING等が指定されていた場合にはこれらを実行木に追加する。この例ではORDER-BYに指定された列を根ﾉｰﾄﾞのK:に設定する(図6ｽﾃｯﾌﾟ7)。

この実行木に対して、木の根から始めて再帰的にﾉｰﾄﾞを辿り、各ﾉｰﾄﾞに対するｺｰﾄﾞを生成する。例えばK:とS:が共に設定されていれば、S:に設定されている選択演算をK:に相当するｿｰﾄ命令の補助機能として実行可能で、この場合には一つのｿｰﾄ命令でK:とS:の処理が可能となる。一方、K:がなく、S:が指定されている場合には新たにｿｰﾄ命令を生成する必要がある。図6に示された実行木に対して得られたｺｰﾄﾞが図5に示されたﾌﾟﾗﾝである。

本ｼｽﾃﾑでは、これらK:、S:、J:の全ての可能な組み合わせに対して生成される命令のﾊﾟﾀﾝを予めｺﾝﾊﾟｲﾗがﾃﾞｰﾀﾍﾞｰｽ化して記憶しており、table drivenな方法によりこれらｺｰﾄﾞ生成を実現している。また、このﾃﾞｰﾀﾍﾞｰｽを変更することによりGREOの命令ｾｯﾄの拡張にも容易に対処することが可能である。

5. おわりに

GREOを用いた試作SQLｼｽﾃﾑの概要と、そのｺﾝﾊﾟｲﾗの実現方式について報告した。本ｼｽﾃﾑは、当社のMELCOM80ｼﾘｰｽﾞでの実用化に向けて更に評価、改良を行う予定である。

[1]安藤他「ﾘﾚｰｼｮﾅﾙﾃﾞｰﾀﾍﾞｰｽﾌﾟﾛｾｯｻGREOの構成」, 電子情報通信学会, DE89-37 1989

[2]伏見他「LSIｿｰﾄﾌﾟﾛｾｯｻ」, 電子情報通信学会, DE88-2 1988

[3]井上他「ﾃﾞｰﾀﾍﾞｰｽﾌﾟﾛｾｯｻRINDAの設計と実現」, 情報処理学会論文誌, 31, 3 1990

[4]JIS ﾃﾞｰﾀﾍﾞｰｽ言語SQL/X3005, 1990

[5]ISO/ANSI,「Database Language SQL2 and SQL3」X3H2-90-001, 1990

[6]Ullmann, J.D.「Principles of Database Systems」, Comuter Science Press 1982