SEQUELについて

渋谷攻昭，鷹尾洋一
（日本IBM Scientific Center）

1 まえがき

データ・ベースの関係形式（Relational Model of Data Base）は、より広汎な末端使用者が、データ・ベースの物理的環境やファイルへのアクセス方法を意識することなく、これらに依存しない言語によって照会を行なうためのモデルである。プログラマー以外の利用者が、ときたまデータ・ベースを利用できるようにするためには、このようなデータ・ベースのモデル化に対応して、照会言語も単純でありなじみ易いものである事が要求される。（照会のための言語は、一般の計算言語に比べて、限られた表現能力しかもたないので、データ準言語 — data sublanguage と呼ばれる。）この方向の努力の一例として、IBM San Jose Research Laboratory において研究開発されているSEQUEL（Structured English QUEry Language）システムについて紹介する。なお、関係形式そのものに関しては、文献［1, 2, 3］を参照されたい。

データ・ベースの関係形式がE.F. Coddにより提唱されて以来、彼自身により、また他の人々により、そのためのいくつかのデータ準言語が提案されてきたし、これからもまた新しい提案がなされるであろう。SEQUELの紹介に入る前に、これらについて簡単に説明しておく。

a)「要素操作型」

PL/I等の汎用言語を親言語として用いられることを目的としており、基本的な少数の機能を用意すればよい。すなわち、1つの関係形式を探査して、与えられた条件を満たすn個組（n-tuple）を逐次取り出す事ができればよい。例としてGAMMA-0［4］があり、また後で紹介するXRMもこの型のinterfaceを提供するものである。

b)「代数型」

照会という操作は、データ・ベース内の関係形式（一般に複数個）から、目的とする関係形式を作り上げる過程である。代数型言語では、和、差、積、直積、射影、結合、商などの集合論的な演算を用いて、関係形式を代数的に結合してゆき、照会を記述する。

c)「写像型」

n項関係は、2項関係の拡張として、そのdomainの任意の部分直積から他の部分直積への写像をすべて表現していると考えることができる。このような写像とand, orにより照会を行なうものである。例としては、SQUARE［5］があり、SEQUELはその変種とみなすことができる。

d)「述語論理型」

検索の条件を述語論理の形式で記述するもので、当然、記述の明確さ、確実さという点では最も秀れている。一般利用者にとっては、必らずしも、なじみやすいものではないが、述語論理で難解なような内容を、他の型で、分かり易く、しかも曖昧さを残さずに表現することも困難であろう。例としては、ALPHA［6］があげられる。

2 SEQUELによる照会 [7]

SEQUELシステムにおける照会の機能を、例を用いて紹介してゆく。
ここで、データ・ベースは次の関係形式から成っているものとする。

```
EMP  (MNO,ENAME,SAL,MGR,DNO)
DEPT (DNO,DNAME,LOC)
```

関係形式EMPは、すべての社員について、社員番号(MNO)、名前(ENAME)、給料(SAL)、上司の社員番号(MGR)、それに所属部門コード(DNO)を与える。また、DEPTは、すべての部門について、部門コード(DNO)、部門名(DNAME)、それに所在地(LOC)を与えるものである。

まず、SEQUELにおける最も簡単な照会例をあげる。

Q1.「所在地がNEWBERGである部門の部門コードを示せ。」

```
SELECT  DNO
FROM    DEPT
WHERE   LOC='NEWBERG';
```

FROM節で検索すべき関係形式名を指定し、WHERE節では検索条件を指定する。また、SELECT節では、検索されたn個組のどのdomainの値を出すか指定する。SEQUELでは、この最も簡単な照会形式をブロックと呼ぶ。

WHERE節には、いくつかの検索条件をAND, ORで結合した形で指定することも可能である。

Q2.「所属部門コードが19で、給料が15000以上の社員名とその上司の社員番号を示せ。」

```
SELECT  ENAME, MGR
FROM    EMP
WHERE   DNO=19
        AND
        SAL>15000;
```

また、WHERE節を省略することもでき、その場合には、関係形式代数の影射(projection)を行なうのと同等になる。

Q3.「社員の表にあらわれる部門コードをすべて示せ。」

```
SELECT  DNO
FROM    EMP;
```

特定のdomainの値ではなく、n個組全体を取り出したい場合には、"SELECT *"という省略表現を用いることができる。

Q4.「給料が20000以上である社員に関するすべての情報を示せ。」

```
SELECT  *
FROM    EMP
WHERE   SAL>20000;
```

SELECT節には、検索結果に操作をほどこすために、SEQUELが用意している組込関数を指定することができる。使用可能な組込関数は、SUM, COUNT, AVG, MAX, MINの5種類である。

Q5.「所属部門コードが17である社員の平均給料を示せ。」

```
SELECT  AVG(SAL)
FROM    EMP
WHERE   DNO=17;
```

WHERE 節に書ける検索条件は、比較演算子を用いた条件とは限らない。ある domain の値が、特定の範囲内にあることを条件にする場合には、WHERE 節に "d BETWEEN v1 AND v2" という形で書くことができる。

Q6.「給料が 10000 から 20000 の範囲内にある社員の名前を示せ。」

```
SELECT  ENAME
FROM    EMP
WHERE   SAL BETWEEN 10000 AND 20000;
```

また、"d IN v" という形で、ある domain の値が、ある集合の要素となっている事を条件として設定することができる。

Q7.「所属部門コードが 17 か 24 か 32 である社員の名前を示せ。」

```
SELECT  ENAME
FROM    EMP
WHERE   DNO IN (17,24,32);
```

この機能を用いると、さらに、下に示すように、ブロックのネストが行なえることになる。

Q8.「所在地が NEWBERG である部門に所属している社員の名前を示せ。」

```
SELECT  ENAME
FROM    EMP
WHERE   DNO IN
          SELECT  DNO
          FROM    DEPT
          WHERE   LOC='NEWBERG';;
```

さらに複雑な照会を行なう場合には、2つの関係形式について、複数個の domain に関して連関(association)をとる必要が生ずる事がある。すなわち、2つの、ブロックの検索条件が互いに他を参照する様な場合である。SEQUEL では、組変数(tuple variable)の導入により、この種の照会も可能にする。

Q9.「給料がその上司の給料以上である社員の名前を示せ。」

```
SELECT  ENAME
FROM    E IN EMP
WHERE   SAL>
          SELECT  SAL
          FROM    EMP
          WHERE   MNO=E.MGR;;
```

Q10.「所属社員数が 10 人以上である部門の部門コードと部門名を示せ。」

```
SELECT  DNO,DNAME
FROM    D IN DEPT
WHERE   10<
          SELECT  COUNT(*)
          FROM    EMP
          WHERE   DNO=D.DNO;;
```

今まで紹介してきた例は、すべて、取り出したい domain がすべて1つの関係形式内にある場合であった。しかしながら、例えば、社員名(ENAME)とその所属部門名(DNAME)が知りたい場合には、ENAME は関係形式 EMP から、また、DNAME は関係形式 DEPT から取り出さなければならない。このように、取り出すべき domain が複数個の関係形式にわたる場合に、SEQUEL では、そのために用意されている COMPUTE 節を用いることになる。

すなわち、COMPUTE節で、下位ブロックから渡される値を変数に与え、この変数名をSELECT節に書くことで、目的が達せられる。

Q11.「社員の名前とその所属部門の名前とを示せ。」

```
SELECT  ENAME,Q
FROM    E IN EMP
COMPUTE Q=
          SELECT  DNAME
          FROM    DEPT
          WHERE   DNO=E.DNO;;
```

Q12.「すべての部門について、部門コード、部門名、それに所属社員の平均給料を示せ。」

```
SELECT  DNO,DNAME,Q
FROM    D IN DEPT
COMPUTE Q=
          SELECT  AVG(SAL)
          FROM    EMP
          WHERE   DNO=D.DNO;;
```

3 SEQUELシステムの内部構造

現在、稼動しているSEQUELプロトタイプ・システムは、IBM・VMF370/CMSの下でのsingle userシステムであり、その概略の構造は、図3.1に示したようになっている。

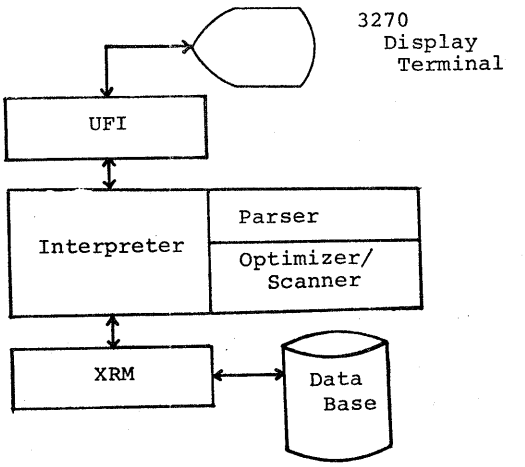

図3.1 SEQUELシステムの構造

UFI (User Friendly Interface)は、端末からの入力あるいは端末への出力を、管理するinterfaceであり、照会の結果得られた表が端末の画面に表示しきれない場合には、scrollingをつかさどる機能ももっている。

interpreterは、UFIから受けとった入力文を、まず構文解析し、得られたparse treeに基づいてデータ・ベースを検索し、結果をUFIに渡す。構文解析には、LALR(k)型のparser [8] が用いられている。なお、XRMおよびoptimizer/scannerについては、次節以降で詳しく述べてゆく。

3.1 XRM (EXtended Relational Memory) [9]

XRMシステムは、任意の次数の関係形式を格納・検索するデータ・ベース・システムであり、SEQUELにおける物理的なデータ・ベース管理および検索は、図3.1に示したように、すべてこのXRMによって行なわれる。また、XRMは、さらに2項関係だけをサポートするRM(Relational Memory)システムの上に組み上げられている。したがって、ここでは、まずRMの紹介を行なう、次いでXRMがRMの上にいかに組み上げられているかを見ることとする。

RMでは、データ・ベース全体を、entity spaceとrelation spaceとの2つに分割し、ともに4kバイトのページ単位で、ディスクとのやりとりを行なう。許されるデータ・タイプとしては、1ワードの数値データと可変長の文字列データとがあるが、文字列データはすべて1つのentityとしてentity spaceに格納され、固有の内部コードすなわち1ワードのeid(entity id)が与えられる。

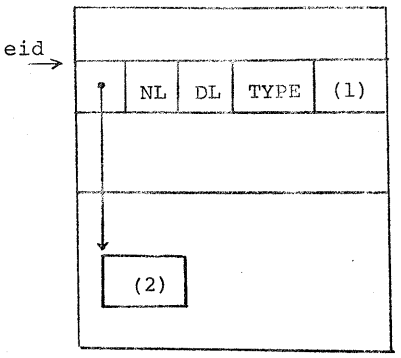

図 3.2 RM entity page format

NL:name length
DL:data length
TYPE:entity type
(1):fisrt 12 bytes of name and data
(2):continuation of (1)

eid=(page no)*c1+(offset)/c2
c1:no of table entries/page
c2:length of table entry

一方、relation spaceには、2項関係が格納され、それらは内部コードrid(relation id)によって識別される。また、これらの2項関係内に現われるデータ項目は、すべて1ワードの数値データもしくは内部コードである。今、ridで識別される2項関係が、$\{ <A_1, B_1>, <A_2, B_2>, <A_2, B_3>, <A_2, B_4>, <A_3, B_5> \}$であるとすれば、relation space内での表現は、図3.3の様になる。

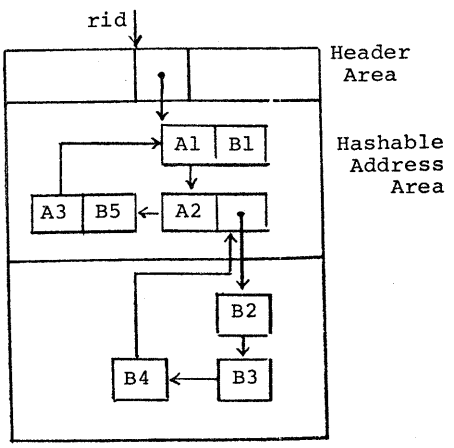

図 3.3 RM relation page format

A1 < A2 < A3
B2 < B3 < B4

rid=(page no)*k1+(offset)/k2
k1:no of header entries/page
k2:length of header entry

rid が与えられれば、header entry の位置が分かり、pointer をたどることで、すべての2個組が得られるし、また、さらに第1 domain の値 Ai も与えられれば、(rid, Ai) を hashing することにより、直接、求める2個組の位置が得られるようになっている。relation page が overflow した場合には、overflow page に2個組が入れられるが、rid でさされる元の page に第1 domain の値についての index を持つために、高々2 page 読むだけで、求める2個組が得られる。

以上述べてきたように、RM における検索等はすべて内部コードに基づいて行なわれる。したがって、検索の時に、文字列データ（外部名）そのものを頻繁に使用したい場合には、使用者自身が index をつくり 文字列から内部コードへの変換を迅速にする必要がある。この index は、文字列 S を hashing して得られる値 h(S) と、S に対応する eid とからなる2個組 <h(S), eid> を要素とする RM の2項関係により実現される。以後、このような2項関係を、特に、hash relation と呼ぶことにする。

XRM は、最初に述べた様に すべての n 項関係を、RM の2項関係によって表現するものであり、RM 同様、処理は内部コードに基いて行なわれる。すなわち、各 n 項関係には rid が、また、各 n 個組には tid (tuple id) が与えられ、その内部コードを指定する事により、検索が迅速に行なえる様になっている。

XRM の関係形式には、大別して、class relation（1項関係）と regular relation（一般の n 項関係）との2種類がある。

class relation は、可変長の文字列データ（外部名）から内部コードへの、あるいはその逆の変換を行なうものであり、RM レベルでは次の様に実現される。

1) hash relation C が作られる。
2) class relation の要素となる各文字列データ S について、
   2.1) S を entity space に登録する。ここで得られる eid が XRM における tid として使われる。
   2.2) S を hashing して得られた値 h(S) と eid とからなる2個組 <h(S), eid> を C に入れる。

regular relation は、RM の2項関係と同じく、1ワードの数値もしくは内部コードだけを、データ項目として含み得る。すなわち、XRM における n 個組は n ワードから成る文字列として表現される。regular relation の RM レベルでの表現は次の様になる。

1) hash relation R が作られる。
2) regular relation の要素となる各 n 個組について、
   2.1) n 個組を n ワードから成る文字列として、entity space に登録する。ここで得られる eid が、XRM の tid となる。
   2.2) n 個組の primary key の値を k とすると、k を hashing して得られた値 h(k) と eid とから成る2個組 <h(k), eid> を、C に登録する。

n 個組を逐次探査ないし直接取り出すためには、普通、tid もしくは primary key の値を指定する必要がある。ただし、XRM では、任意の domain について index ―― これを inversion relation と呼ぶ ―― を作製する様指定する事ができ、これにより、その domain の値から tid を直接得ることができるようになる。inversion relation は、指定された domain の値と、対応する n 個組の tid

とを関連づけるRMの hash relation で表現される。ただし、inversion relation はあくまでも、システム内部で保守・使用されるものであり、XRM使用者が直接その中味を取り扱うことは許されない。
XRMシステムにおけるデータ・ベース・カタログは、master relation と呼ばれ、やはり関係形式になっている。master relation は、データ・ベース内の各n項関係に対応したn個組を要素としており、domain, inversion relation, primary key等々に関する情報をもっている。なお、master relation におけるtidが、XRMにおける対応するn項関係のridとなっている。

3.2 Optimizer / Scanner

検索の最適化とは、本来、ディスクへのアクセス回数を最小にすることであるが、SEQUELでは、XRMおよび仮想記憶システム(VMF370は仮想記憶システムである。)のpagingを制御することが不可能であるため、その目標を、n個組を取り出す回数を最小にするという点に置く。言い換えれば、index(inversion relation)を極力利用して、逐次探査をする範囲を可能る限り小さくすることである。例えば、"domain-name = value"という条件が与えられた場合、indexが利用可能であれば、ただちにtidのlistを得ることができる。このようなindexが利用可能な条件に注目して、最終結果に入り得るn個組の範囲を せばめ、そのtidのlistを作成するのがOptimizerであり、このtidのlistに基づいてn個組を読み込んでcheckをし、最終結果を作りあげるのがScannerである。Optimizer / Scannerによる処理は、A, B, Cの3段階に分かれており、以下、順次それらについて述べてゆく。なお、照会文は一般に、ブロックの木構造をなしており、全体としての処理はその木構造の上から下へと再帰的に行なう。

A) ブロック内の各条件を次の5つに分類する。

P1: indexの利用により ただちに解決できる(tidのlistが得られる)もの。すなわち、"domain-name = value"または"domain-name IN value-list"の形をしているもの、あるいは下位ブロックを含んでいても、そのブロックが単純で上と同等とみなせるもの。例えば、DNOのindexがある時、
DNO IN SELECT DNO FROM DEPT WHERE LOC='NEWBERG'。

P2: indexが存在しないか、あるいは、存在してもそれを利用できないもの、たとえば、DNO > 27。ただし、上位ブロックと、組変数による関連はもたないものであり、したがって、1回の逐次探査により、解決できるものである。

P3: 上位ブロックと、組変数による関連をもつが、上位から1つのn個組が与えられれば、indexにより解決できるもの。例えば、EMPのDNOについてのindexがある時、

```
SELECT NAME FROM D IN DEPT WHERE 10>
        SELECT COUNT(*) FROM EMP WHERE DNO=D.DNO;;
```

。

P4: P3と同じだが、ただindexが存在しないもの。この場合、必要であれば、一時的なindexが作製される。

P5: 上位ブロックと、組変数による関連があり、しかもindexが存在しても、それを利用する事ができないもの。

A段階が終わって、ブロックに P1, P2 だけしかない場合には B段階へ進み、そうでない場合には C段階へ進む。

B) この段階では、各条件は P1 か P2 のいずれかであり、P1なら index によって解決可能(R)、P2なら不可能(N)の印をつける。一般に、検索条件全体は、AND, OR で結ばれた木構造をしているので、更にその木全体を渡り歩いて、すべての node に R か N の印をつける。と同時に、条件の list である P*_list を作ってゆく。(Index Selection Algorithm)

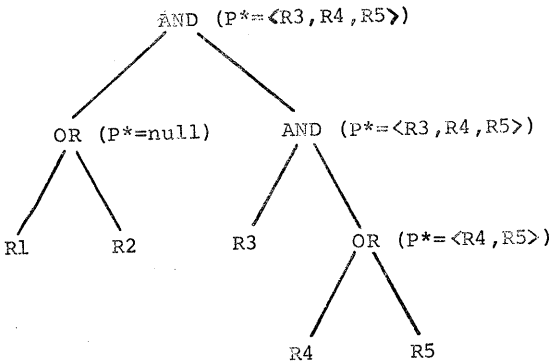

図 3.4
Index Selection の例

例から分かるように、木構造の根に対応する P*_list は、index によって、あらかじめ解決することが有効である条件の list となっている。

つぎに、P*_list にもとづいて、index を利用しながら tid の list (Li) を各 node について作成してゆく。このとき、index で完全に解決されたものには R を、最終結果に入り得る n 個組の範囲がせばめられた場合には S を、また、それ以外の場合、すなわち最終結果の候補となる n 個組が依然としてその n 項関係全体にわたっているものには N の印をつける。(List Combining Algorithm)

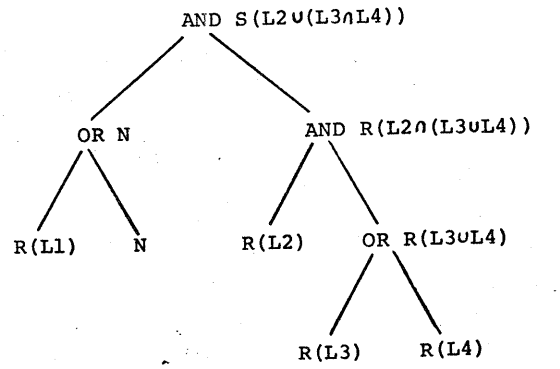

図 3.5
List Combining の例

Scanner は、この木構造に基づいて n 個組をとり出し最終結果をつくりあげる。もし、この木構造の根が R であれば、単に tid の list に対応する n 個組をとり出すだけである。S または N であれば、n 個組を 1 個 1 個取り出して、条件を満たすかどうか check する。(Test Tree Algorithm)

c) この段階で処理されるブロックは、少なくともP3, P4, P5のうちの1つを含むものであり、上位ブロックと、組変数により関連づけられている。したがって、このブロックは、上位ブロックから渡されるn個組1個1個に対応して、反復処理をしなければならない。
そこで、反復処理における手数を出来るだけ減らすために、次の様な前処理を行なっておく。まずIndex Selection Algorithmにより、P*-listを作り上げる。ただし、このときP1, P2, P3, P4ならばRとし、P5ならばNとする。次にこのP*-listに現われるすべての条件に対して、その条件のタイプに応じて次の様な処理を行なう。

P1: List Combining Algorithmにより tidのlistを作る。
P2: Scannerによる逐次探査と条件のcheckを行ない、P1と同様に最終的なtidのlistを作る。ただし、もしこのP2ブロックが、自分自身の組変数と関連を持つような下位ブロックを含むならば、Scannerを再帰的に呼び出して、その下位ブロックの処理も行なってしまう。
P3: 何もしない。
P4: 一時的なindexを作り、P3に変換する。

以上の前処理が終わった段階で、Optimizerは、その上位ブロックの処理にもどる。ただし、このとき、このブロックは反復処理が必要であるという事を示しておく。
なお、反復処理の際には、まずP3に対してList Combining Algorithmにより tidのlistを作成し、それからTest Tree Algorithmにより、最終結果の作成に移る。

参照文献

[1] Codd,E.F., A Relational Model of Data for Large Shared Data Banks, Comm. ACM 13(1970) 377.

[2] Codd,E.F., Relational Completeness of Data Base Sublanguages, Courant Computer Science Symposia,Vol.6,Prentice Hall(1971).

[3] Codd.E.F., Fuether Normalization of the Data Base Relational Model, Courant Computer Science Symposia,Vol.6,Prentice Hall(1971).

[4] Bjorner,D.,Codd,E.F.,Deckert,K.L. and Traiger,I.L., The Gamma-0 N-ary Relational Data Base Interface Specifications of Objects and Operations, Research Report RJ 1318,IBM Research Laboratory,San Jose(1973).

[5] Boyce,R.F.,Chamberlin,D.D.,King,W.F. and Hammer,M.M., Specifying Queries as Relational Expressions,Proc. of ACM SIGPLAN/SIGIR(1973).

[6] Codd,E.F., A Data Base Sublanguage Founded on the Relational Calculus, Proc. of ACM SIGFIDET Workshop(1971).

[7] Chamberlin,D.D. and Boyce,R.F., SEQUEL: A Structured English Query Language,Proc. of ACM SIGFIDET Workshop(1974).

[8] Lalonde,W.R.,Lee,E.S. and Horning,J.J., An LALR(k) Parser Generator, Proc. of IFIP Congress 1971,Nrth Holland(1972) 513.

[9] Lorie,R.A., XRM--An Extended(N-ary) Relational Memory,G320-2096, IBM Scientific Center,Cambridge(1974).

附録：SEQUEL の構文法

A) データ・ベース照会

t-query ::= SELECT {name, ···} FROM [var-name IN] tab-name
*

[WHERE sel-exp] [COMPUTE var-name = {val-exp, ··}];
s-query
t-query

s-query ::= SELECT {COUNT, AVG, MAX, MIN, TOT} ({[UNIQUE] col-name})
*

FROM [var-name IN] tab-name [WHERE sel-exp];

sel-exp ::= sel-term を AND, OR, (,) を用いて組み合わせたもの

sel-term ::= {
val-exp θ {val-exp, s-query, t-query}
col-name IN {t-query, (literal, ···)}
< col-name, ··· > IN {t-query, (< literal, ···>, ···)}
val-exp BETWEEN val-exp AND val-exp
}

val-exp ::= col-name, var-name.col-name, literal に関する算術演算式

θ ::= { =, ¬=, <, <=, >, >= }

B) データ・ベース管理

UPDATE [var-name IN] tab-name SET col-name = {val-exp, s-query}, ··· [WHERE sel-exp];

INSERT INTO tab-name [(col-name, ···)] : {< literal, ···>, t-query};

DELETE [var-name IN] tab-name [WHERE sel-exp];

EMPTY TABLE tab-name;

CREATE TABLE tab-name col-name (dom-name), ··· {KEY IS col-name, KEYS ARE (col-name, ···)};

DROP TABLE tab-name;

CREATE DOMAIN dom-name ({NUM, CHAR});

DROP DOMAIN dom-name;

{CREATE, DROP} INVERSIONS ON tab-name (col-name, ···);

ASSERT assert-name ON [var-name IN] tab-name : sel-term;

{DROP, DISPLAY} assert-name;