数理解析研究所講究録
第525巻 1984年 78-99

分散型関係データベースシステムにおける テーブルの分割と重複

図書館情報大学図書館情報学部
増永良文

1. はじめに

分散型関係データベースシステムでは，リレーションテーブル（以後単にテーブルとよぶ）がサイト間に"効果的"に分散しているという保証は一般にはなく，テーブルを分割（水平，垂直）あるいは重複して生成して，システム全体としてより効果的なデータベース運用を図ることが期待できる。現在分散型関係データベースシステムの周知例としてはシステムR*[1]，分散型INGRES[2]他幾つかの例を挙げることができるが，テーブルの分割・重複を導入したシステム構成とはなっておらず，上記問題は大事な問題とは認識されてはいるものの手つかずの状態である。理論的には多重コピーを生成した場合の更新シンクロナイゼーションをどうとるかという問題はインプリメンテーションというが幾つかの論文で論ぜられているが[3]，具体的に重複テーブルをどう管理・運用する

か，分割テーブルをどう管理・運用するかについては，ほとんど解析例が報告されていない。

本稿の目的は分散型関係データベースシステムにテーブルの分割と重複を導入することの定義，管理・運用上の問題点をかいつまんで議論することにある。本問題は検討すればする程奥が深いことが判る。例えばインデックスや権限付与の問題等も十分見直を要求される。テーブルの分割と重複を許さない場合のシステムアーキテクチャはそれを許した場合，根底的に見直を要求されることになると理解してよいと思う。本稿では議論の対象としなかったが，テーブルの分割と重複を許した場合の分散質問処理体系は本質的にそうでない場合に比べて，別物となろう。我々はまず想定する分散型関係データベースシステムの概略を述べることから始める。

2. 協調的分散型データベースシステム

分散型データベースシステムは種々の目的で構築される。また採用されるアーキテクチャも目的に依り多岐にわたる。大別すると集中型のアーキテクチャ（たとえば集中システムカタログ，集中トランザクション管理，集中デッドロック検出・解消，等を）とるシステムと非集中型のアーキテクチ

をとるシステム構成には大別されよう。さらに後者は質問処理技術の観点から大別して(イ)完全分散方式と(ロ)協調的分散方式の二つに分離できるように思われる。前者は分散システムを構成する各サイトが同じアルゴリズムを実行するが複雑で時間のかかるネゴシエイションを必要とする。後者は適宜質問処理のコオーディネータを決め，他サイトはそれに協力して，質問処理を行なってゆこうとするものである。後者は前者に比べて各サイトの自治权が尊重され，異なったバージョンでもシステム全体が稼動しうる等のメリットが考えられる。リレーショナルで均質で後者に属するシステムの実際構成例としてシステムR*が挙げられよう[1]。システムR*のアーキテクチャの詳細は本稿では触れないが，本稿ではこのフレームワークの中でテーブルの分割・重複を議論してゆくことにする。

3. テーブルの分割と重複の定義

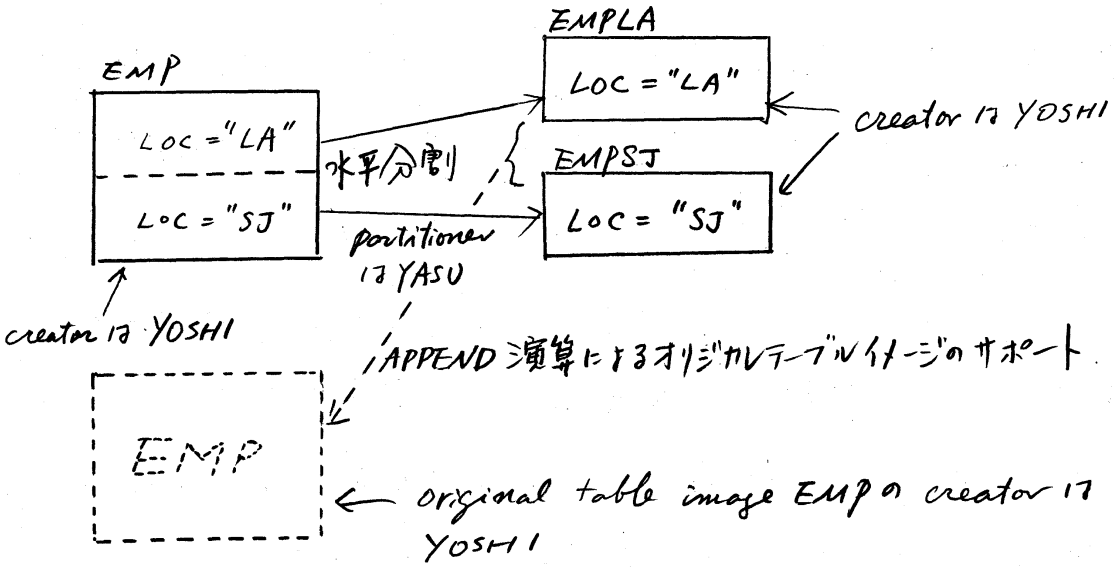
テーブルの分割と重複の直感的定義は明らかだろう。分割には水平分割と垂直分割の二つを考える。水平分割されてえられたオリジナルテーブルの水平部分テーブルを水平フラグメントということにする。もしテーブルTが$T_1, T_2, \cdots, T_n$に水平分割されれば各フラグメントはどれも素

である。 例論T = T1 APPEND T2 APPEND・・・APPEND Tn である。 ここにAPPENDはタップルの重複を保存したまま和集合をとる演算である。 UNIONではなくAPPENDが必要な理由はシステムR*で許すSQLテーブルでは一般にタップルの重複が許されているからである。APPENDの詳細は他稿をみられたい。(4) 水平分割のSQLステートメント表現は既にアイデアが発表されている(1)のでそれにならい下記のようにしよう。

DISTRIBUTE TABLE <table-name> HORIZONTALLY INTO
<fragment-name> WHERE <search-condition>
[IN dbspace-name @ site]
:
<fragment-name> WHERE <search-condition>
[IN dbspace-name @ site]

ここに[ ]はオプショナルである。 なお本稿で想定している協調的分散型関係データベースシステムでは，システムのデータオブジェクト(基底リレーション，ビュー，フラグメント，コピー等)は creator @ creator-site . object-name @ object-birth-site の4項組からなるシステムワイド名(SWN)を持つので，水平フラグメントについてもそれを明確に定義しておかないと，アクセスお果たくなる。

いまテーブルTのcreatorをuser1，creator-siteをsite A，Tのbirth-siteをsite B，Tの水平分割要求者をpartitioner1，そのsiteをsite Cとするとき，水平フラグメントのcreator及びcreator-siteはuser1とsite Aとする。そのbirth-siteはTのbirth-siteであるsite Cとする。勿論フラグメントはユニークなシステムワイド名を付けられる必要がある。水平フラグメントの格納サイトはIN句で指定される。デフォールトは原テーブルと同じサイトの同じdb spaceとする。Tのcreatorであるuser1とpartitioner1の間には本章では言及しないが，Tの分割をuser1の代りにpartitioner1が行なってもよいという認定をuser1が行なったという権限付与の体系の存在を仮定している。partitioner1ではなくuser1がフラグメントのcreatorになっていると定義した理由は，オリジナルテーブルのcreatorはuser1であり，partitioner1はあくまでuser1がuser1の代りにテーブルを分割してもよいと認めた一partitionerにしか過ぎないことという権限付与のながれに依る（第8章参照）。オリジナルテーブルイミージ（文献(4)で導入している，OTIと略する）のcreatorはuser1である。この間の関係を図-1に示す。（creatorはownerと読みかえてもよいだろう。）



図－1．　フラグメントの creator の定義関係

次に垂直分割について記す。まず SQL に準じた構文を示す。

DISTRIBUTE TABLE <table-name> VERTICALLY INTO

<fragment-name><column-name-list> [IN dbspace-name @ site]

⋮

<fragment-name><column-name-list> [IN dbspace-name @ site]

テーブル T(X)，ここに X は column-name-list とする，とし，T を $T_1(X_1)$，…，$T_n(X_n)$，ここに X_1 ～ X_n は column-name-list とする，に垂直分割したとする。X_1 から X_n は X の部分集合であるが，一般にそれらを勝手に選んだのでは，垂直分割は information-lossless でなくなる。現在の可リレーションの情報無損失分解の仮定かつ

十分条件に垂直フラグメントからの原テーブルの復元操作にどのような演算体系を用いるかに依存して，多値従属性（MVD）で決まる場合と，ジョイン従属性（JD）で決まる場合とが知られている。関数従属性（FD）はMVDの特別な場合で，情報無損失分解のための十分条件をなす。我々の垂直分割をどの範囲で捉えるかは想定される各種応用により決まることとなろう。本稿で立ち入る問題ではない。

垂直フラグメントのシステムワイド名の命名法は水平フラグメントのそれに全く準ずるとする。ただし，オリジナルテーブルイメージのサポートにどのような演算体系が仮定かは上記分解のポリシーに依存する。MVDの範囲内においては自然結合演算（タプルの重複がおきる可能性のあるSQLテーブルではタップル重複保存自然結合演算[4]）で十分となる。

続いてテーブルの重複について述べる。SQLに準じた重複定義の構文は次のとおりとする。

DISTRIBUTE TABLE <table-name> REPLICATED INTO
<replica-name> [IN dbspace-name @ site]
⋮
<replica-name> [IN dbspace-name @ site]

レプリカは各々異なったシステムワイド名を持たねばならな

いことは言を待たない。レプリカのcreator，そのsite，birth-siteはフラグメントの場合に準ずる。フラグメントやレプリカのbirth-siteをオリジナルテーブルのbirth-siteのそれと同じくする意味は，そうしておくとオリジナルテーブルもフラグメントもレプリカも全て同じサイトの一つのシステムカタログ中に登録されることになり，単一カタログテーブルの参照によりオリジナルテーブルの分割・重複構造が判るように出来るからである。

最後にテーブルの分割や重複を御破算にする概念について議論しておく必要がある。個々の分割や重複を一つ一つとりあげそれをなかったことにするという概念は勿論有るが，それを指定するステートメントも複雑となるし，またそれを実現するカタログ管理方式も複雑になることが予想されるので，そのような概念はとり下げ，オリジナルテーブルを一挙に復元してしまう方式を考える。もし既にある分割や重複の一部だけを修正したかったのであれば，それは残念ながら一度元に戻ったオリジナルテーブルから希望の分割・重複の構造となる様それを行なってくれと要求することである*。このためのSQLに準じた構文を次のようにする。

REBUILD TABLE <table-name>

---

*ただしREBUILD文とフラグメント・レプリカレベル迄許す拡張は容易である。

4. テーブルの分割と重複操作の制御

フラグメントを分割・重複の対象とするか，レプリカを同様それらの対象とするか，あるいは一度分割されてしまったテーブルのオリジナルテーブルイメージをその対象とするかはシステムのテーブル分割と重複の管理・運用体系の根幹にかかわる問題で十分明確にしておかなくては分らない。

この問題に関しては色々な立場があろうが，本稿で想定する制御体系は次のとおりであり，それに批判，検討を加える形でこの問題の認識を深めたい。

(a) 任意のフラグメントは分割と重複の対象とする。

(b) 任意のレプリカは分割と重複の対象とする。

(c) オリジナルテーブルイメージは分割と重複の対象としない。

つまり，極言すれば一度分割されてしまったテーブルはもう分割や重複の対象とはなりえるけれども，そうでないものは分割でも重複でもいづれの対象にもなりえるという極めて自由度の高い制御体系を想定している。しかしこの際問題になるのは次のような場合であろう。たとえばアプリケーション1のテーブルTを水平にT_{H1}とT_{H2}に分割すると効率良く処理でき，アプリケーション2はTを垂直にT_{V1}とT_{V2}に分割すると効率良く処理出来るというような，コンフリク

トした分割基準(criteria)が与えられた場合にシステムはどう対処するかということであろう。一案は T を例えば T_{H1} と T_{H2} に分割して，更に T のオリジナルテーブルイメージに水平分割を施して T_{V1} と T_{V2} を作り，二つの要求を同時にアコモデイトするということであろうが，この方法は上記(c)項により不可能である。しかしこの場合見捨てないで，先ずは T のレプリカ TREP1 と TREP2 を作り，TREP1 を水平分割して T_{H1} と T_{H2} を作り，TREP2 を垂直分割して T_{V1} と T_{V2} を作るという便法でそれを克服することにする。色々な要求が出てきた場合，前章の REBUILD TABLE ステートメントを駆使して，レプリカ生成作戦をとることになるかもしれないが，解決はできるという立場に立つ。

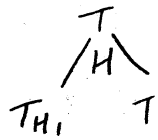
なおフラグメントやレプリカを分割したり重複したりする人はそれなりの特権(privilege)を持たなければならない。その体系については次8章で概略を述べる。

5. テーブル分割・重複構造の表現

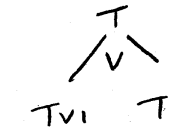
前節までのフラグメントとレプリカのチューニング法および制御法のもとで，テーブルの分割と重複の構造がロジカルにどのように表現できるかを本章で考察する。考察の軸となる二つの概念はその木表現と線型性である。(5)

まず木表現について述べる。

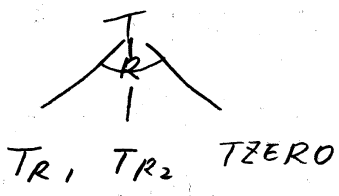
いまテーブル T が T_{H1} と T_{H2} に水平分割されたとする。このとき，この分割を次の様に木表現する。



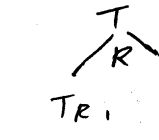
もしテーブル T が T_{V1} と T_{V2} に垂直分割されたとする。このとき，この分割を次の様に木表現する。



もしテーブル T からレプリカ T_{R1} と T_{R2} が作られたとする。このとき，この重複を次の様に木表現する。

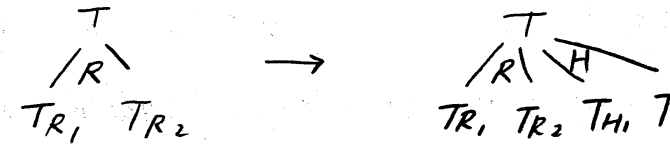


こて，ここで木表現の線型性に言及しなければならない。もし水平分割や垂直分割の木表現方式にしたがうなら，テーブルの重複は下図のように書き表わされるべきなのであろう。

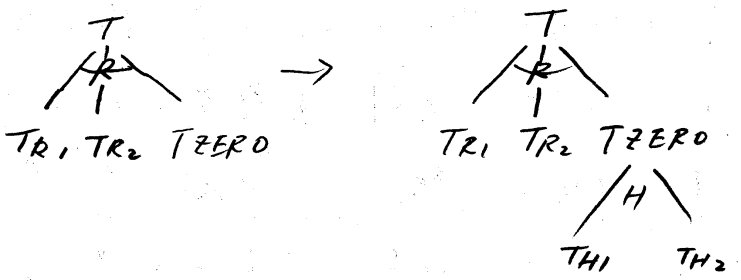


しかし本章で想定したように，レプリカは作られたものの，原テーブルは分割と重複の対象となりうる。したがって，

もしテーブル T が新たに分割・重複，たとえば T を T_{H1} と T_{H2} に水平分割するという分割の対象となったとき，それをアコモデイトしようとすると，重複の後者の木表現の場合，



とでも表現するより他に，表現の仕様がない。しかしこの表現は，左辺の木が右辺の木の葉ノード（leaf node）からの水平分割が表現できていないという意味で，線型（linear）ではない，と定義する。一方重複の前者の定義に従うならば T の水平分割は TZERO の水平分割と実質的に読み替えて，次の様に線型表現できることになる。なお，



TZERO は T のレプリカの一つであって，T から T_{R1} と T_{R2} を重複せよと特権をもったユーザに指定されたときに，システムが自動的に生成するレプリカである。このレプリカの生成者及び生成タイミングはそのユーザのそれと，格納される dbspace 及びタイミングはオリジナルテーブルのそれと全く一致させる。TZERO の実際の生成については次章の下

ーブル分割と重複のカタログ管理の章で具体的に述べる。

6. テーブルの分割・重複管理のためのカタログ構造

協調的分散型リレーショナルデータベースシステムは，各サイトがそのサイトで誕生あるいは格納されているデータオブジェクト（テーブル，フラグメント，レプリカ等）の情報を記録，保持するためのカタログ（一つのリレーション）を持つ。このカタログをシステムR*にならいシスカタログ（SYSCATALOG）と呼ぼう。

このSYSCATALOGにテーブルの分割と重複の情報を記録する。まずテーブルTがT$_{H1}$とT$_{H2}$に水平分割されたとする。このとき，Tの他にT$_{H1}$とT$_{H2}$が次の様にエントリーされる：

エントリーT：　分割・重複タイプ＝"H"

エントリーT$_{H1}$：　親テーブル名＝"T"
　　格納サイト名＝"xxx"

エントリーT$_{H2}$：　親テーブル名＝"T"
　　格納サイト名＝"YYY"

実際にこのサイトに格納されているテーブル，フラグメント，あるいはレプリカにはTABID（TABle IDentifier）が付く。データオブジェクトの格納サイトのSYSCATALOGにはそのオブジェクトがここでもエントリーされる。ここ

では具体的にこのオブジェクトに関する格納の統計情報（たとえばそのテーブルのタップルの平均長，タップル統計量）も記録する。レがレプラグメントの誕生サイトをオリジナルテーブルの誕生サイトと一致させているため，オリジナルテーブルの誕生サイトのSYSCATALOGテーブルのみを見ることにより，そのテーブルの分割（そして重複）構造を一度に読みとれるようにしていることに注意する。

次にテーブルが垂直に分割されたときのデータオブジェクトのエントリーを定義する。これは水平分割と変らないが，分割・重複タイプ＝"V"とする。

テーブルの重複について述べる。重複の場合，分割・重複構造の線型性を保つために一工夫する。この概略は前章終りに述べた。いまテーブルTからレプリカTR1とTR2が定義されたとする。Tの誕生サイトのSYSCATALOGの構造を定義する。

(1) Tの誕生サイトがその格納サイトでもあるとき：

エントリーT：　TABID＝"0"
　　分割・重複タイプ＝"R"

エントリーTZERO：TABID＝"重複される前のTのTABID"
　　親テーブル名＝"T"，システム生成＝"Y"
　　格納サイト名＝"自サイト名"

エントリー TR_1 ： 親テーブル名 = "T"
格納サイト名 = "xxx"
エントリー TR_2 ： 親テーブル名 = "T"
格納サイト名 = "yyy"
が SYSCATALOG に登録される。

(2) Tの誕生サイトと格納サイトが異なるとき：

[Tの誕生サイトの SYSCATALOG]
エントリー T： 分割・重複タイプ = "R"
エントリー TZERO：親テーブル名 = "T"，システム主キー = "Y"
格納サイト名 = "Tの格納サイト名"
エントリー TR_1：親テーブル名 = "T"
格納サイト名 = "xxx"
エントリー TR_2：親テーブル名 = "T"
格納サイト名 = "yyy"

[Tの格納サイトの SYSCATALOG]
エントリー T： 分割・重複タイプ = "R"
TABID = 0
エントリー TZERO：親テーブル名 = "T"
TABID = "重複される先のTのTABID"

テーブルの重複の際，その分割・重複構造を線型に保つ為に導入された TZERO テーブルは，実際にはTのレプリカを作

或してYれを定義する必要はなく，まずTZEROという名前のみ登録し，次いでYのTABIDをTのそれに設定し，TのTABIDを0（零，これはこのテーブルがこのサイトの実リレーションでないことを表わすとする）に設定すれば実現できることに注目する。したがってYの作業量は程知れている。もしYの後あるユーザからテーブルTを分割あるいは重複したいという要求が出されたならば，Tは実リレーションでないので受けつけない。（もしTの代りにTZEROを生成した責任をシステムがとり，めんどうをみるならば，親テーブル名＝"T"でかつシステム生成＝"Y"なるレプリカを探し出しYれを分割あるいは重複の対象と自動的にするというメカニズムも考えられようが，煩雑である。）ユーザは分割や重複に先立って，たとえばPRSTRUCTURE〈テーブル名〉のようなユーティリティコマンドを用意しておき，指定したテーブルの分割・重複構造を視認できるようにしておき，それを参照して，常に分割・重複構造木の葉（leaf）を新しい分割あるいは重複の対象とするようしたちがよい。その時システムが生成したTZEROのようなテーブルにはテーブル名の右肩に例えば"*"印でもつけてそれをユーザに知らしめるように工夫する。

このようにTZEROの導入をはかれば，一度分割あるいは

重複されたテーブルは二度と分割あるいは重複の対象となることはなく，これを一重分割－一重重複環境と呼ぶことにする。(5)

7. テーブル分割・重複とインデックス

リレーションテーブルのインデックスはまたリレーションテーブルである。インデックスはテーブルの格納サイトに格納されている。

さてテーブルTがTH1とTH2に水平分割されたとき，もしTにインデックスIが定義されていたとする。まずTはTH1とTH2に分割されてしまい，最早実体のないテーブルとなったので，Iは意味がない。*-2 水平フラグメントTH1とTH2のインデックス付に対して少なくとも二つのアプローチがあろう。（Iを属性(の組)X上のインデックスとする。）

(1) TH1とTH2共にX上のインデックスを作成する。

(2) TH1とTH2にはその生成時にとくにインデックスは付けない。もしインデックスを付けたいのなら（特権をもったユーザが）その都度付ける。

(1)の案は親テーブルに付いていたインデックスを子テーブルが素直に受け継ごうという発想である。分割したイン

*-2 Iをどう処分するかは別の問題である。

インデックステーブルをフラグメントに付随して送り（あとにTABIDの調整が必要となろう）それを実現する方法と，X上でインデックスを付けるべきというSQL文をフラグメントに付随して送る方法等が考えられよう。しかし代替案(1)は何故Tが分割されねばならなかったかを考えると，時にはおかしいことをやっていることになるかもしれない。つまり，TH_1ではX上のインデックスを重宝するアプリケーションが多いが，TH_2ではY（X≠Y）上のインデックスを重宝するアプリケーションが多いから，分割要求が発生したかも知れないのである。そのような時には(2)の方が自然である。

次にTがT_{V1}とT_{V2}に垂直分割されたとき，やはり少くとも二つの代替案が考えられよう。

(1) T_{V1}の属性集合がXを含めば，X上のインデックスを作成する（T_{V2}についても同様）。

(2) T_{V1}とT_{V2}にはその生成時にとくにインデックスは付けない。もしインデックスを付けたいなら（特権をもった ユーザが）必要に応じその都度付ける。

代替案の選択にあたっては，原則的に水平分割のところで述べたと同じ議論が成立しよう。

最後にTがT_{R1}とT_{R2}に重複されたとする（システムはそれに伴ないレプリカTEERDをも生成したとする）。この と

き，Tのインデックス工は自動的にそのまま TZERO のインデックスとなる。 TR1 と TR2 のインデックスの定義にも分割と同様の二つの代替案があろう。

(1) TR1 と TR2 には工と同じインデックスを生成する。

(2) TR1 と TR2 にはその生成時にとくにインデックスを付けない。 もしインデックスをはりたいならば特権をもったユーザが必要に応じてその都度付ける。

同じタイプのアプリケーションが多数Tに集中するのでTを重複生成したのであれば(1)の代替案が有力であろう。そうでなければ(2)かもしれない。どのようなアプリケーションが想定されるかにより，案が決まろう。この アプリケーション依存の特質は分割，重複いずれの場合にも案決定の重要な因子となろう。この戦略は質問処理体系と共に相まって解決してゆくべき問題である。

8. テーブル分割・重複の権限付与

フラグメントやレプリカの生成者はオリジナルテーブルの生成者であると3章で定義した。あくまでテーブルの partitioner や replicator はそのテーブルの creator から分割することや重複することのみの特権を付与されたにすぎないという認識であった。こうすることにより，テーブル

の生成者はいつでも簡単な権限付与スキームのもとで，分割をキャンセルしたりレプリカをとりつぶしたりできる。ここでテーブル（フラグメントやレプリカを含む）を分割したり重複したりできる特権（privilege）をDISTRIBUTION特権ということにする。テーブルの生成者はそのテーブルに対するこの特権を有する。他にDBA（DataBase Administrator）はこの特権を有する。他ユーザへのDISTRIBUTION特権の付与は，RETRIEVEやUPDATE等に対するGRANTと同じ形とする。

さて，あるユーザがテーブルTに特権P（たとえば，SELECT，INSERT，DELETE，UPDATE，ALTER，INDEX等）を持っているとする。またTが分割，あるいは重複されたとする。我々はこのユーザに，Tの全てのフラグメント，及びレプリカ上でPを有すると想定する。

ここで，テーブルのpartitionerやreplicatorをフラグメントやレプリカの生成者にはしなかった（オ3章）理由を述べる。テーブル（フラグメントやレプリカを含む）の生成者（creator）はそのテーブルに対して全ての特権を持つと考えるのは極めて自然である。そこでテーブルTにSELECT特権は持つが，UPDATE特権を持たないようなユーザUに，DISTRIBUTION特権を与えてしまったとする。

すると，ユーザ1は丁の水平フラグメントを作れば，そこで UPDATE 特権を持つことになり，許されていなかった丁の update を実質的に行なえるようになる（これを権限付与スキームだけで検出禁止しようとすると，"このユーザには UPDATE を禁止する"という禁止条項を陽に指定し，フラグメント上で特権が付与していることを検出する方法が考えられようが，こうするとリレーション検証に手間どることが予想される他，本来性質の異なった DISTRIBUTE と例えば UPDATE という二つの特権が partitioner が creator に与えられるという変な構により概念に混同をきたしてしまうことになる。）

9. おわりに

本報では物論的分散型関係データベースシステムにテーブルの分割と重複を許すようにした場合のデータオブジェクト管理上の諸問題を議論した。この結果，その一管理体系が明らかになった。この体系は特に分割と重複に強い制限を加えるものではなく，自由度の大きいものと思う。

今後，このようなデータオブジェクトの管理体系のもとで，分散質問処理方式をどう設計するか，あるいは最適分割・重複の問題等が議論されねばならない。

[謝辞] 本研究に御関心を示して下され，御討論，御批判下さった諸氏に深謝いたします。

なお本研究は昭和58年度文部省科学研究費補助金の交付をうけているので付記いたします。

[文献]

(1) R. Williams et., "R*: An Overview of the Architecture", Proc. Intl. Conf. on Database Systems, Jerusalem, pp.1-27 (1982).

(2) M. Stonebraker et., "A Distributed Database Version of INGRES," Proc. 2nd Berkeley Workshop on Dist. Data Management and Computer Networks, pp. 19-36 (1977).

(3) P.G. Selinger, "Replicated Data", in Distributed Data Bases, I.W. Draffan et. ed., Cambridge Univ. Press, pp. 223-232 (1980).

(4) 増永，"分散型関係データベースシステムにおけるテーブルの分割と重複の導入，" 情報処理学会第27回(昭58年後期)全国大会，6K-4，pp. 751-752 (1983)

(5) 増永，"分散型関係データベースシステムにおける分割・重複テーブルの一管理方式，" 情報処理学会第28回(昭59年前期)全国大会，6E-2，発表(1984)