

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の一般にはなく、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開発例として1)システムR^{*}、分散型ENGRES²⁾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いない。上記問題は大事な問題と認識されているものの手つかずの状態である。理論的には複製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して幾つかの論文で論じられているが、³⁾ 具体的には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、ほとんど解析例が報告されている。

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。本問題の検討すればする程深が深いことが判る。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した場合のシステムアーキテクチャはそれを許した場合、徹底的に見直を要求されることになると思う。本稿では議論の対象とてはなかったが、テーブルの分割と重複を許した場合の分散型処理体系の本質的にどうであるかの場合に比べて、別物となる。我々に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。また採用す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、集中データロック検出・解消、等々）をとするシステムと非集中型のアーキテクチャ

をとるシステム構成には大別出来る。さらに後者は値間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と考えられる。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か複雑で時間のかかるシミュレーションを必要とする。後者は適宜値間処理のコーディネータを定め、他サイトと協力して、値間処理を行なっているものとする。後者の前者に比べて各サイトの自治権が尊重され、異なったバージョンでもシステム全体が稼働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いるシステム 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られませんが、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・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区別する。水平分割と称して与えられたオリジナルテーブルの水平部分をテーブルを水平フラグメントとすることにする。もしテーブル T が $T_1, T_2, \dots, T_n$ に水平分割されれば各フラグメントは互いに素

である。 句読 $T = T_1 \text{ APPEND } T_2 \text{ APPEND } \dots \text{ APPEND } T_m$ である。 ニニは APPEND は タップル の重複を保存した集合をとる演算である。 UNION で付る APPEND が付る理由はシステム R^* で許可 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 の詳細は他稿とみられた⁽⁴⁾。 水平分割の SQL ステートメント表現の際に ⁽¹⁾ アイテムが各表に与えられているので以下に以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 ]
```

ニニは [] はオプションである。 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、ユーザ等)は $\text{creator} @ \text{creator-site} . \text{object-name} @ \text{object-birth-site}$ 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これを明確に定義しておかると、アクセスが楽になる。

いるテーブル T の creator は user 1, creator-site は site A , T の birth-site は site B , T の水平分割要求者は partitioner 1, T の site を site C とすると、水平フラグメントの creator は creator-site は user 1 と site A とする。 T の birth-site は T の birth-site である site C とする。 勿論フラグメントの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 user 1 と partitioner 1 の間には本章では言及しないが、 T の分割を user 1 の代りに partitioner 1 が行なってもよいという認定を user 1 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。 partitioner 1 である user 1 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 user 1 であり、partitioner 1 はあくまで user 1 が user 1 の代りにテーブルを分割してもよいと認めた partitioner にか過ぎないことという権限付与の仕組みに依る(第8章参照)。 オリジナルテーブルイメージ(文列(4)で差入している、OTI と略する)の creator は user 1 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。)

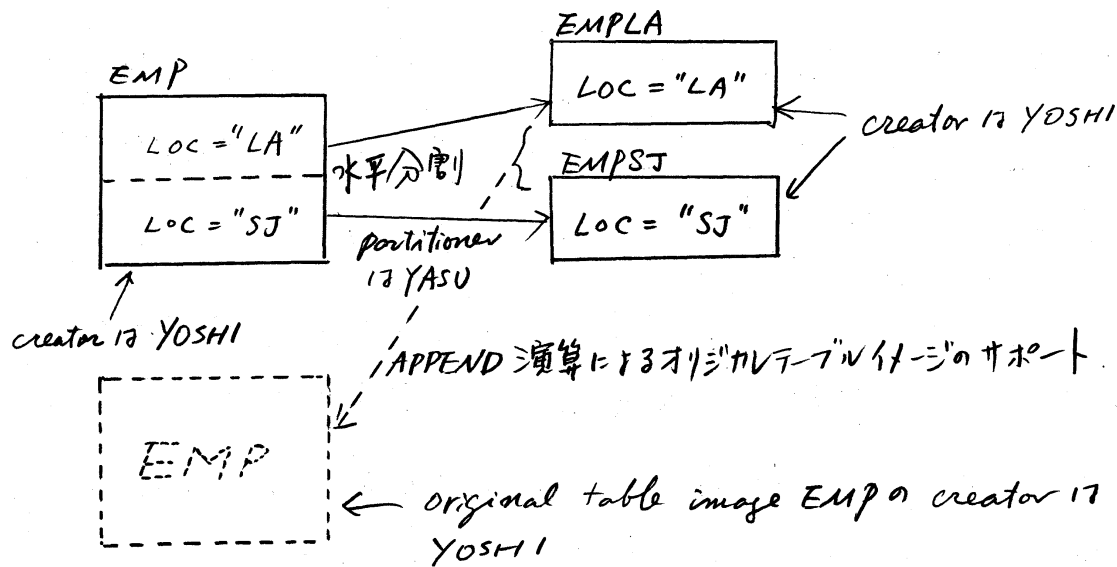


図-1. フラグメントの Creator の定義関係

次に垂直分割について記す。まず SQL に準いた構文を示す。

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 $T(X)$, 二重に X は column-name-list とする,

とし, T を $T_1(X_1), \dots, T_n(X_n)$, 二重に $X_1 \sim X_n$

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。現在のリレーションの情報損失分解の必要条件

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(M
 VD)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。皮相従属性(FD)はMVDの特殊な
 場合で、情報無損失分解のための十分条件を与える。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
 り決まると考えよう。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な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必要かは
 上記分解のポリシーに依存する。MVDの範囲内であれば
 自然結合演算(テーブルの重複がなくなる可能性のあるSQL
 テーブルでのタ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 dlspace-name @ site]
  :
  <replica-name> [IN dlspace-name @ site]
```

レプリカは各々異なるシステムワイド名を持たねばならぬ

いことは言を得た。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, そうしてあ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あり, 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取りあげそれを消したことにするという概念は勿論存するが, それを指定するステートメントも複雑となるし, 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, 今のような概念のとり下り, オリジナルテーブルを一言に復元してしもう方式を考へる。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, それを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うべくと要求することである*。このための SQL に導い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可拡張は容易である。

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、レプリカを同様こ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なくてはならない。

この問題に関して色々な立場があるが、本稿で想定する制御体系は次のとおりであり、これに批判、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、要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見做りえる。けれども、そうであるものの分割でも重複でもいつれの対象にも見做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。しかしこの除肉題にあるのは次のような場合である。たとえばアプリケーション1のテーブルTを水平にTH1とTH2に分割すると効率良く処理でき、アプリケーション2のTを垂直にTV1とTV2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)がなかった場合には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と例えばTH1とTH2に分割して、更にTのオリジナルテーブルに水平分割を施してTV1とTV2を作り、二つの要求と同時にアクセス可能であるということであるが、二のちねは上記(c)項に則して可能である。しかしこの場合見捨てられて、代わりにまずTのレプリカTREP1とTREP2を作り、TREP1を水平分割してTH1とTH2を作り、TREP2を垂直分割してTV1とTV2を作るという便法でこれを克服することにする。色々の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、レプリカ生成作業をとることにするかもしられるが、解決はできるといふ立場に立つ。

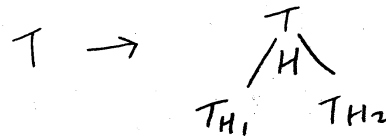
各みフラグメントやレプリカを分割したり重複したりする人は与えられた特権(privilege)を持たなければならぬ。この体系については第8章で概略を述べる。

5. テーブル分割・重複構造の表現

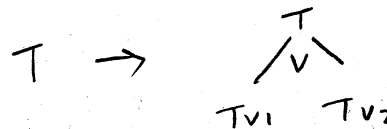
前節までのフラグメントとレプリカのネーミング法とび創制法のもとで、テーブルの分割と重複の構造がロジカルにどのように表現できるかを本章で考察する。考察の軸となる二つの概念のその本表現と線型性である⁽⁵⁾。

まず木表現について述べる。

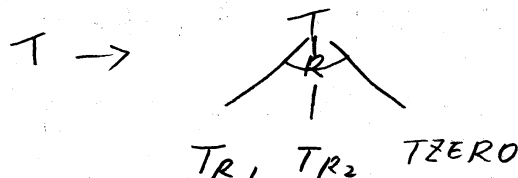
もしノード T が TH_1 と TH_2 に水平分割されたとする。このとき、この分割を次の様に木表現する。



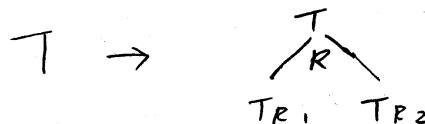
もしノード T が TV_1 と TV_2 に垂直分割されたとする。このとき、この分割を次の様に木表現する。



もしノード T からレプリカ TR_1 と TR_2 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。



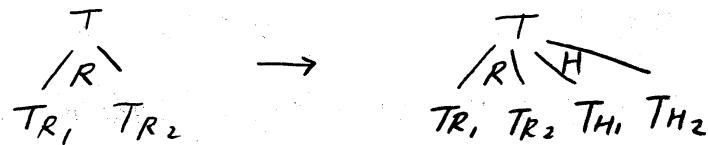
さて、ここで木表現の線型性に言及しなくてはならない。もし水平分割や垂直分割の木表現を仮にしていたら、ノードの重複は下図のように書き表わさなければならないであろう。



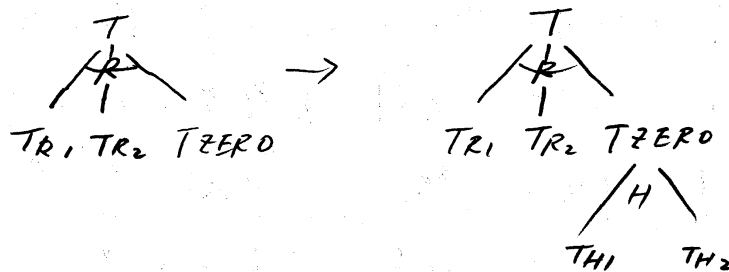
しかし本章で規定したように、レプリカは作られたものの、元ノードは分割と重複の対象となりうる。したがって、

“

もしテークル T が新たな分割・重複, たとえば $T \in TH_1$ と TH_2 に水平分割するとこの分割の対象となったとき, それをアクトゥエイトしようとするとき, 重複の後者の木表現の場合,



とでも表現する。他に, 表現の仕様が異なる。しかしこの表現は, 左辺の木が左辺の木の葉ノード (leaf node) からの水平分割が表現できている。という意味で, 線型 (linear) である, 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 TR_1 と TR_2 と重複せよと指令をもったユーザにのみ指定されたときには, 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者サットはユーザのユーザ, 格納される $dbspace$ 及びサットはオリジナルテークルのユーザと全く一致する。 $TZERO$ の実際の生成については次章の T

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^* における「メタカタログ」(SYSCATALOG) と呼ぶ。

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。データベースオブジェクトの格納サイトの SYSCATALOG にはこのオブジェクトが二つでもエントリーされる。二つ

では具体的にはこのオブジェクトに関する格納の統計情報(たとえばテーブルのクワールの平均長, クワール総数等)も記録する。しかしフラグメントの誕生サイトも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 JYSCATALOG テーブルのみに見ることにし, このテーブルの分割(そして重複)構造も一度に読みとめ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。この水平分割と変化するが, 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 TR_1 と TR_2 が定義されたとする。 T の誕生サイトの JYSCATALOG の構造を定義する。

(1) T の誕生サイトがこの格納サイトでもあるとす:

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-TR₁ : 親テーブル名 = "T"
格納サイト名 = "XXX"

エントリ-TR₂ : 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場合 :

[T の誕生サイトの SYSCATALOG]

エントリ-T : 分割・重複タイプ = "R"

エントリ-TZERO : 親テーブル名 = "T", システム名 = "Y"
格納サイト名 = "T の格納サイト名"

エントリ-TR₁ : 親テーブル名 = "T"
格納サイト名 = "XXX"

エントリ-TR₂ : 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-T : 分割・重複タイプ = "R"
TABID = 0

エントリ-TZERO : 親テーブル名 = "T"
TABID = "重複される前の T の TABID"

テーブルの重複の際, Y 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

成してそれを定義する必要はなく、まず「TZERO」という名称のみ登録し、次いでその 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二のサイトの実行レーションで二を二と表わすとする) に設定すれば実現できることに注目する。したがってその作業量は極めて小さい。もしそ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 は実行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それを分割あるいは重複の対象と自動的に行う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、それを登して、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にした方がよい。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 "※" EP でマークしてそれをユーザに知らしめるように工夫する。

二のように TZERO の導入をばかれば、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...は重複の対象となる
 こととなる、これを「一重分割」-「一重重複環境」と呼ぶこととする。⁽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ネチクレーション
 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 TH_1 と TH_2 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
 TH_1 と TH_2 に分割されてしまい、最早実体のないうテーブルと
 なったので、 I は意味がなくなる。^{*-2} 水平フラグメント TH_1 と TH_2
 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
 る。 (I を属性(の組) X 上のインデックスとする。)

- (1) TH_1 と TH_2 共に X 上のインデックスを作成する。
- (2) TH_1 と TH_2 には X の生成時とくにインデックスは付
 けられない。もしインデックスを付けたらなら (特権をもち
 たユーザが) X 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ぐという発想である。分割したイン

^{*-2} I をどう処分するかは別の問題である。

ディクショナリをフラグメントに付随して送り(あとに
TABLEDの調整が必要となる)やれを実現する方法と、X上
でインデックスを付与すべきという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が分割されたか知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、TH1ではX上のインデックスを重宝するアプリケーション
があるが、TH2ではY($X \neq Y$)上のインデックスを重宝す
るアプリケーションがあるから、分割要求が発生したかも知
れないのである。今のところ時には(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Yの生成時にすでにインデックスは付
けられている。もしインデックスを付与しないなら(特権を持った
ユーザが)必要に応じて再度付ける。

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Y
に伴ってレプリカTEEROを生成したとする)。このとき

主、 T のインデックス I は自動的に γ の $TZERO$ のインデックスとなる。 TR_1 と TR_2 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) TR_1 と TR_2 には I と同じインデックスを生成する。

(2) TR_1 と TR_2 には γ の生成時と同じにインデックスを付ける。もしインデックスを付けたら (特権を持ったユーザが) 必ずに γ の更新を行う。

同じタイプのアプリケーションが多数 T に集まるので T を重複生成したのであれば (1)の代替案が有力であらう。 γ うでなければ (2)がもしれる。どのところアプリケーションが想定されるかにより、案が決まらう。このアプリケーション依存の性質は分割、重複いづれの場合にも案決定の重要な因子となる。この戦略は質問処理体系とも相まって解決してやぐべき問題である。

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する章で定義した。あくまでテーブルの *partitioner + replicator* は γ 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過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。そ
こでテーブル（フラグメント+レプリカ+合成）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 与 UPDATE 等
に対する GRANT 形式と同一とする。

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）
を持ってゐるとする。もし 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 を有すると規定する。

ここで、テーブルの partition + replicator をフラ
グメント+レプリカの生成者にはしるかった（オラクル）理由
を述べる。テーブル（フラグメント+レプリカ+合成）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 特権を持つが、UPDATE 特権を持たないような
ユーザ U に、DISTRIBUTION 特権を与えてしまつたとする。

すると、ユーザ7はTの半々フラグメントを作れば、YでUPDATE 特権を持つことになり、許されているかつTのupdateを実行時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き、"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予備していることを検出する余裕が考えられようが、こうするとリッチな検証に手間がかかることが予想される他、本来性質の異なったDISTRIBUTEと例えればUPDATEという二つの特権がpartitioner or creatorに与えられるという変な橋により概念に混同をまねてしまうことになる。）

9. おわりに

本稿では協調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、分散型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るであろう。

[謝辞] 本研究に御関心を示して下さい、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ゐるので付記いたします。

[文献]

- (1) R. Williams 他, "R*: An Overview of the Architecture", Proc. Intl. Conf. on Database Systems, Jerusalem, pp.1-27 (1982).
- (2) H. Stonebraker 他, "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, Z.W. Drazfman 他 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パーティションの分割と重複の導入," 情報処理学会第27回(昭和58年春期)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・重複パーティションの管理方式," 情報処理学会第28回(昭和59年春期)全国大会, 6E-2, 登録予定(1984)