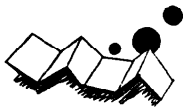
解　説



属性モデルを用いたデータベースシステム†

東　田　正　信††

1. はじめに

近年のデータベース技術の進歩は目覚ましく，ユーザのデータベースシステムへの関心も高まって来ている．このような状況の下で本誌では「データベース技術」の特集[1]が組まれ，その基礎技術から応用技術まで幅広く紹介されている．

データベースの処理系において最大の問題はいわゆる“90—10ルール”[2]という言葉に表現されるように，大量のデータの中から要求に合った一部のデータだけをいかに効率よく取り出すかということである．すべてのデータベースシステムはこの目標を達成することを最大の目標としているといっても過言ではない．このためにデータ表現のためのデータモデル，データベースを格納するファイルの管理方法などのデータベース構築技術から分散型データベースシステムやデータベースマシンに見られるようなシステム構成技術やハードウェアによる処理技術に至るまで各方向からのアプローチが試みられている．

本稿では，これらのアプローチの中から複数バックエンド型データベースシステムを取り上げる．このシステム構成の特長は，要求される処理性能に柔軟に対応できるということである．一方，データベースの分割・格納方法，分散されたデータベースの維持管理方法などの問題点もある．ここで紹介する属性ベースデータモデル*（以後“属性モデル”）はこれらの問題をうまく処理できるとして提案され，現在このモデルを用いた複数バックエンド型データベースシステムMDBS（**M**ultiple-backend **D**ata **B**ase **S**ystem）が米国オハイオ州立大学で試作・開発が進められている[3)~6)]．

筆者は1981年9月から一年間このプロジェクトに参加する機会を得た．ここでは属性モデルの概要および複数バックエンドシステムの環境下での同時実行制御の手法について，MDBSでの実現例を引用しながら解説する．

2. 複数バックエンド型データベースシステム

複数バックエンド型データベースシステムとは単一のコントロールプロセッサに複数のデータベース専用のバックエンドプロセッサが接続されたシステムであり，その構成は種々のものが考えられるが，例としてバス構成の場合を示すと**図-1**のようになる．

複数バックエンド型データベースシステムに要求されることは，全体としての性能を損うことなくデータベースの大規模化が図れ，かつ単位時間に処理できるリクエスト数を増加させることである．このためには以下の事項を満足することが必要である．

（1） システムのスループットがバックエンド数に比例して増加する．

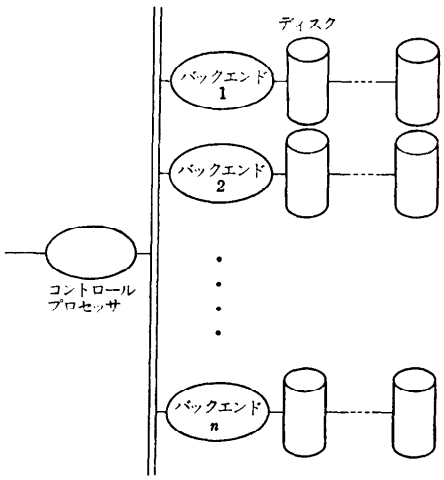


図-1 複数バックエンド型データベースシステム構成例

† An Implementation of a Multi-Backend Database System Using Attribute-based Data Model by Masanobu HIGASHIDA (Yokosuka Electrical Communication Laboratory, N. T. T.).

†† 電電公社横須賀電気通信研究所

* attribute-based data model

（2） レスポンスタイムがバックエンド数に反比例して向上する．

（3） データベースの規模の増大，システム性能の向上などの要求に応じてシステムの拡張が可能である．これに伴うハードウェア，ソフトウェアの改造・新規ハードウェアの追加が不要である．また新しいプロセッサが付加されたことによるシステム全体への影響がない．

このようなシステムを実現するためには以下に述べるようなことがポイントとなる．

① データベースはすべてのバックエンドで均等に分割されて格納される．このためには，データベースの分割が容易であること，バックエンド間でのデータベースの維持管理・無矛盾性の保証が容易にできることなどが必要である．

② すべてのバックエンドはリクエストを効率よく処理することができる．これには1つのリクエストの処理に必要な作業が複数のバックエンドで分散処理できること，同時に処理可能なリクエスト数を増やして並列処理の度合を上げることなどが必要となる．

ここで紹介する属性モデルと呼ばれるデータモデルは，これらの要求条件に合致するものとして取り上げられた．

3. 属性ベースデータモデル[2),7)]

3.1 データモデルの概念

属性モデルは Hsiao らによって提案され[7)]，その後オハイオ州立大での DBC[8),9)] への採用が検討されたデータモデルである．しかし DBC の構想はまだ具体化されていないため，MDBS での採用が実際のシステムへの適用例としてははじめてのことである．

属性モデルは従来の階層型・網型・関係型のデータモデルのいずれとも異なる概念に基づくが，データの捕え方としては，関係型に最も近い．関係モデルがタプル (tuple) を基本要素と考えるのに対し，属性モデルでは**キーワード** (keyword) と呼ばれる属性と属性値の対が基本要素となる．ここで属性とはファイル名，タイプ，性質，特徴などである．タプルに相当するものは属性モデルでは**レコード** (record) と呼ばれ，キーワードの集合として定義される．（属性モデルの特徴は何と言っても，この基本概念のシンプルさにある．このシンプルさが後に詳述するデータベースのクラスタ化による分割・高度な並列処理を可能にしている．）

キーワード ＝〈属性，属性値〉

レコード ＝（キーワード，キーワード，……）

（レコード例）

（〈ファイル名，従業員〉*，〈職位，マネジャ〉

〈所属，おもちゃ売場〉，〈給料，30,000ドル〉）

キーワードはその用途に応じて以下の3種類がある．

- シンプルキーワード……レコード検索に用いる．
- セキュリティキーワード…アクセス制御に用いる．
- ディレクトリキーワード……クラスタ作成（**3.3** 参照）に用いる．

次に，クラスタ化の道具となる**述語** (predicate)**，**記述子** (descriptor) を定義する．まず，述語とは，

述語 ＝（属性，比較演算子***，属性値）

で表わされる属性と属性値の関係を定義するものである．与えられたキーワードは，その属性が述語を規定する属性と同一で，その属性値が述語で規定される属性と属性値との関係を満足する時，その述語を満足しているという．例えば，〈給料，15,000ドル〉というキーワードは，述語（給料＞10,000ドル）を満足していると言う．

記述子は述語または述語の組合せによって定義されるもので，属性値や属性値の範囲で規定された属性として表現される．

$$\text{記述子} = \left\{\begin{array}{l}(\text{述語}) \\ \text{or} \\ (\text{述語}) \wedge (\text{述語})\end{array}\right\}$$

（記述子の例）

① （給料≧20,000ドル）∧（給料≦30,000ドル）

または（20,000≦給料≦30,000）

ここで定義される記述子をクラスタ化に使用するためには互いに排他的でなければならない．その意味は①のように定義される記述子間でその値の範囲にオーバラップがあったり，②で定義される記述子は①で定義される記述子と重複してはならないということである．これは各々のレコードがただ一つのクラスタに属するために必要な条件である．

最後にデータベースへのアクセスの手段を提供するものとして，**問合せ連結** (query conjunction) と**問合せ** (query) を定義する．問合せ連結は，述語の連結として，また問合せは任意の述語のブール表現とし

---

* 通常，最初のキーワードは，<ファイル属性，ファイル名>を表わす．

** 正確にはキーワード述語 (Predicate keyword)．

*** 比較演算子とは ＝，≠，≧，＞，≦，＜のうちの1つである．

て表わされる．以下に例を示す．

① 問合せ連結の例

(給料≧25,000ドル)∧(所属＝おもちゃ売場)∧
(名前＝Hsiao)

② 問合せの例

((所属＝おもちゃ売場)∧(給料<10,000ドル)∨
((所属＝本売場)∧(給料>20,000ドル))

以上が属性モデルのデータベースに関する主な概念である．

3.2 データ操作言語 (DML*)

属性モデルデータベースシステムで使用するデータ操作言語 (DML) は，他システムと同様4種類のリクエスト――検索 (retrieve)，挿入 (insert)，削除 (delete)，更新 (update)――をサポートする非手続き型の言語である．これらのリクエストの形式および例を以下に示す．

(1) 検索リクエスト (RETRIEVE)

(形式) RETRIEVE (問合せ), (目標リスト), 〔BY-属性〕

ここで目標リストとは出力項目のリストを表わしBY-属性は，リストの出力範囲を限定するのに用いる．

(例1) RETRIEVE (ファイル名＝従業員)
∧(所属＝おもちゃ売場), (名前)

(例2) RETRIEVE (ファイル名＝従業員)
∧(給料>10,000ドル), (名前, 給料)

(例3) RETRIEVE (ファイル名＝従業員),
(給料の平均値), 〔BY―所属〕

(2) 挿入リクエスト (INSERT)

(形式) INSERT (レコード)

(例) INSERT (〈ファイル名, 従業員〉,
〈給料, 13,000ドル〉, 〈所属, おもちゃ売場〉)

(3) 削除リクエスト (DELETE)

(形式) DELETE (問合せ)

(例) DELETE (名前＝Hsiao)∨(給料＝50,000ドル)

(4) 更新リクエスト (UPDATE)

(形式) UPDATE (問合せ) 〈修飾子**〉

(例) UPDATE (ファイル名＝従業員)
〈給料＝給料＋5,000ドル〉

(1)～(3)までは，従来のDMLと比較して大きな差異はないが，更新リクエストでは修飾をうける属性によって処理の仕方が異なる．これは後に述べるクラスタ化と深い関係がある．更新リクエストではある属性値の値が変更されたり，属性そのものが変化したりする場合*がある．属性モデルでは記述子を用いて定義したクラスタによってデータベースを管理するが，更新された属性値が，そのレコードの所属するクラスタを定義している記述子で規定されている値域をはずれる時には所属すべきクラスタを変更する必要が生ずる．属性自体が変化する場合もそれがクラスタ定義に関わっていれば同様の処理を必要とする．

3.3 クラスタ化とデータアクセスの手法

クラスタ化とは指定されたレコードを指定されたキーワードをもとに分類することである．クラスタ化の目的は2つある．

(1) データベースを分割し，複数のバックエンドに分散して格納する．

(2) データアクセス時に，探索領域を小さくしリクエストを満足するレコードを検索する時間を最小にする．

クラスタ化の手法によって，システムは高いスループットと短いレスポンスタイムを実現することが可能となる．

クラスタは同一の**記述子セット**から成るレコードの集合として定義される．記述子の中に含まれる属性(これを**ディレクトリ属性**と呼ぶ)を持つキーワードを**ディレクトリキーワード**と呼ぶ(**3.1**参照)が，一般に一つのレコードは複数のディレクトリキーワードを持つ．これらのキーワードから生成される記述子の集合を記述子セットと呼ぶ．同一の記述子セットに属するレコード群がクラスタを形成する．すべてのレコードがこの方法によりただ一つのクラスタに属する**．(この際に**3.1**での記述子の定義での排他性が重要な役割りを果たす．)

実例によりこのプロセスを説明する．まず記述を簡単にするために，各々の記述子に一つの識別番号を

表-1 記述子―記述子ID対応表 (DDIT)

記　述　子	記述子ID
給料≦20,000ドル	D 1
20,000<給料≦40,000	D 2
40,000<給料≦60,000	D 3
20≦年齢≦30	D 4
31≦年齢≦50	D 5
51≦年齢≦70	D 6
性別＝女	D 7
性別＝男	D 8

* Data Manipulation Language

** 修飾をうける属性および属性値が，更新後にどのように変化するかを記述する．(例参照)

* 例えば<修飾子>が<月給＝年俸/12>の形で与えられた時など．

** 論理的な説明については参考文献2)を参照されたい．

表-2　入力レコードの例

レコード番号	レコードの内容	対応する記述子IDのセット
R1	<給料, 25,000><年齢28><性別男>	{D2, D4, D8}
R2	< 〃 10,000>< 〃 23>< 〃 男>	{D1, D4, D8}
R3	< 〃 13,000>< 〃 33>< 〃 女>	{D1, D5, D7}
R4	< 〃 15,000>< 〃 35>< 〃 女>	{D1, D5, D7}
R5	< 〃 45,000>< 〃 45>< 〃 女>	{D3, D5, D7}
R6	< 〃 17,000>< 〃 25>< 〃 男>	{D1, D4, D8}
R7	< 〃 19,000>< 〃 27>< 〃 男>	{D1, D4, D8}
R8	< 〃 50,000>< 〃 50>< 〃 女>	{D3, D5, D7}

表-3　クラスタ定義表 (CDT)

クラスタ番号	記述子セット	レコード
C1	{D2, D4, D8}	R1,
C2	{D1, D5, D7}	R3, R4
C3	{D1, D4, D8}	R2, R6, R7
C4	{D3, D5, D7}	R5, R8

付与する DDIT 表 (Discriptor-to-Discriptor ID* Table) を作成**する. (表-1)

ここで 表-2 のレコード入力が与えられたとすると, これに対応する記述子 ID のセットは 表-1 を用いて 表-2 の右端に示すように表現することができる. このセットがクラスタを定義し, 表-3 に示すクラスタ定義表 (CDT***) ができる.

このようにしてクラスタ化されたレコードはクラスタ単位ごとにバックエンドが指定され, 分散して格納される. 属性モデルを用いるとデータベースをレコード単位で扱うことができ, クラスタ化の手法を用いて処理の分散化が図れ, さらに並列性の向上を実現することができる.

次に, こうして作成されたデータベースへのアクセス方法について述べる. ある問合せに対応するレコードを求めるためには, そのレコードを含むクラスタ (番号) を探し出す必要がある. 例として次の問合せを考える.

RETRIEVE (15,000≦給料≦25,000) ∧
(20≦年令≦30) ∧ (性別=男)

これに対応する記述子の組合せは, 表-1 により,

(D1 ∨ D2) ∨ D4 ∧ D8

となる. これを書換えると

{D1 ∧ D4 ∧ D8} ∨ {D2 ∧ D4 ∧ D8}

となって探索すべきクラスタ番号は表-3 を参照して C1 および C3 と決めることができる. (実際にはクラスタ定義表は使わないで, この逆表 (記述子—クラスタ対応表) を作成しておき, これを用いる.) こうして求めたクラスタ中のレコードはすべてもとの問合せを満足するとは限らないので, 1つ1つのレコードをもとの問合せと照合して満足するレコードを選び出す. この例では, C1, C3に含まれる4つのレコード R1, R2, R6, R7 がチェックの対象 (探索領域) となるが, もとの問合せに適合するのは R1, R6, R7の3レコードである.

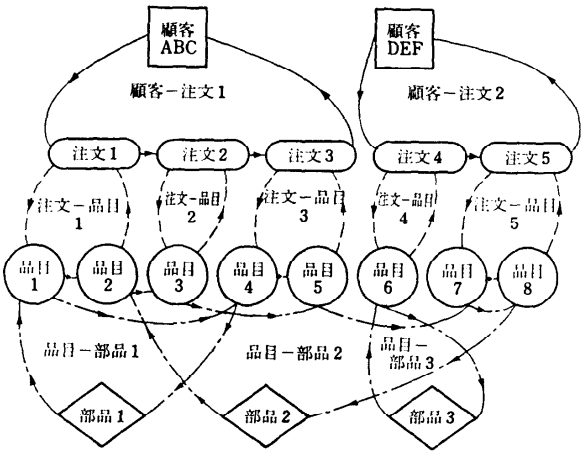


図-2　網型データベースの例

* Identification
** これはデータベース入力前に, ユーザが記述子を定義する形で実行される.
*** Cluster Definition Table

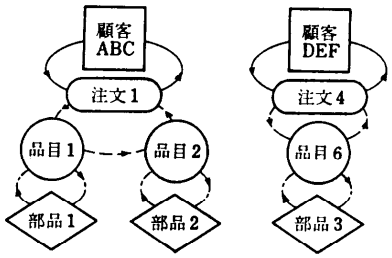
3.4　他モデルとの比較

前節までにおいて, 属性モデルの概要・特徴をのべたが, ここでは他モデルとの比較を行う. 主なクライテリアとして, データベースの分割性, 処理の並列性, 従来モデルからの移行性の3項目を取り上げる.

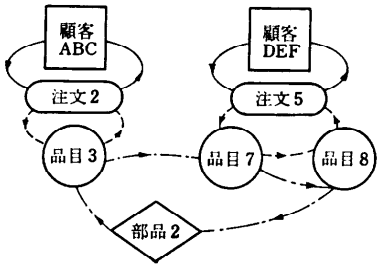
(1)　データベースの分割性

例えば 図-2 のような網型データベースの例を考えてみる.

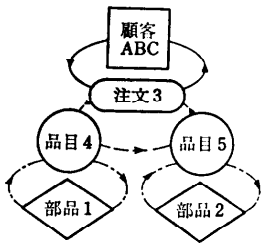
この例では4つのレコードタイプ (顧客, 注文, 品目, 部品), 3つのセットタイプ (顧客—注文, 注文—品目, 部品—品目) から成るが, 検索時間短縮のために, 例えばこれを3つのバックエンドに分割格納しようとすると, 図-3 のようになる. これからもわかるように, データに冗長性が生じこれがメモリの非効率化, 更新時のデータの無矛盾性の保持などの処理の煩雑化などが生じる. (これは, 網型モデルでは, エンティティはレコー



バックエンド 1



バックエンド 2



バックエンド 3

図-3　図-2 のデータベースの3つのバックエンドへの分割例

ドで表現され，関係（リレーション）はポインタで表現されるという2つの異なるメカニズムが混在していることに起因する）．こうした理由から網型，階層型のデータモデルは，ここで対象としている分散処理システムには適していないことがわかる．一方，属性モデルでは，すべての論理的な概念（エンティティやリレーション）が属性と属性値のペアとして表現されるため，冗長度なく分割可能となりこのようなシステムに適合する．

（2）　データベース処理の並列性

データベースが分散格納されていても，これを操作する言語 DML がデータ処理の並列性を許していないとこれを活かすことができない．階層型のデータベース（例えば IBM のIMS など）では DML* はデータを逐次的に，同時には一レコードの処理方法をとる傾向にあり，このような方式ではこのシステムの特質を活かすことができない．属性モデルシステムでは，一つの問合せは述語のブール表現で表わされるため，これをいくつかのリクエストに分解し，各々が要求するクラスタを並列にアクセスすることができる．更に後述する同時実行制御を行い，並列度をあげる方法を取り込むことができる．

（3）　従来のデータベースからの移行性

現在ある大多数のデータベースシステムは，網型，あるいは，階層型のデータモデルをサポートしているが，RDBM[10]，DIRECT[11] などの分散型データベースシステムは関係型データモデルをサポートしている．このため新しいシステムへの移行を考えると，これらデータモデル間でのデータの変換，問合せの変換などが必要となるが，これらは決して容易ではない．Banerjee[12]らは，DBC の検討段階で，階層型，関係型のデータベースおよび問合せの属性型への変換が容易にできることを示している．例えば SEQUEL の問合せを属性型の問合せに変換する例を示す．

（例）

SEQUEL：

```
SELECT NAME
FROM EMP
WHERE DNO=50
```

属性モデル DML：

```
RETRIEVE ((Filename=EMP) ∧
  DNO=50)), (NAME)
```

このように属性モデルのデータベースシステムでは他モデルとのインタフェースを作成することにより比較的容易に他のデータベースモデルもサポートすることができる．

このように属性モデルは，分割性，並列性，移行性などの観点からみて，複数バックエンド型のデータベースに適したモデルであると言うことができる．

4. 同時実行制御*

複数のバックエンドから成るシステムでは並列に実行できるリクエストの数がシステムの性能を大きく左右する．ここで重要なことは，複数のリクエストを実行する前と後とでデータベースに矛盾が生じないように実行の順序を制御することで，これを同時実行制御と呼んでいる．この制御方式として最もよく用

* IMS では DL/I と呼ばれている．

* Concurrency Control

いられているのは**二相ロック方式***であるが，その実現にあたっては種々のメカニズムが考えられている．

ここでは，複数バックエンドシステムにおいて無矛盾性を保証するとはどういうことなのか，また同時実行制御の実現にあたってシステムのオペレーティングシステムがロック制御方式にどのように関わってくるのかについて説明し，次いで MDBS での実現例を紹介する．

4.1 無矛盾性の保証

無矛盾性には，**データ内無矛盾性 (inter-consistency)** と**システム内無矛盾性 (intra-consistency)** の2つがある．これらを説明する前に**互換可能**なリクエスト (compatible request) と**交換可能**なリクエスト (permutable request) の定義を行っておく．

- 互換可能リクエスト (compatible request)

リクエスト $r1$ および $r2$ の実行においてこれら2つのリクエストを同時に実行する**時に得られる結果が，$r1$ を先に $r2$ を後に実行した場合およびこの逆の順に実行した場合のいずれとも等しい（すなわち，実行の順序によらない時）にこれらのリクエストは互換可能であるという．この場合にデータ内で矛盾が生じる可能性のあるリクエストは互換不可能 (imcompatible) であるという．例えば，2つの検索リクエストは互換可能であるが，以下に示すように2つの更新リクエストは互換不可能***である．

(互換不可能な例)

$r1$: UPDATE (ファイル名＝従業員)
〈給料＝給料＋2〉

$r2$: UPDATE (ファイル名＝従業員)
〈給料＝給料×2〉

- 交換可能リクエスト (permutable request)

互換不可能なリクエストであっても，すべてのバックエンドでその実行順序を守らなくてもよいときに，これらのリクエストを交換可能であるという．(即ち $r1 \rightarrow r2$ の順にリクエストを実行した時の結果と $r2 \rightarrow r1$ の順にリクエストを実行した時の結果が等しい．) 一般に2つの削除，検索，挿入リクエストは交換可能（もちろん，互換可能）であるが，その他の組合せは交換可能ではない*．

(交換不可能な例)

$r1$: DELETE (給料＞50,000)

$r2$: RETRIEVE (給料＞25,000) (by 年令)

以上からわかるように互換可能であるためには少なくとも交換可能であることが必要である．また互換不可能なリクエストによる矛盾は単一バックエンドシステムでも複数バックエンドシステムでも起こりうる．

さてデータ内無矛盾性の保証とは，同時実行できない（互換不可能な）リクエストの順序性を守ることによって，複数リクエストの実行後の無矛盾性を保証することである．次のシステム内無矛盾性の保証とは，複数バックエンドシステムで必要となるものであり，交換可能でないリクエストの順序性を守ることによって，全バックエンド間でのデータの無矛盾性を保証することである．

一般に複数バックエンド型のデータベースでは，このシステム内無矛盾性を保証することは，すべてのバックエンド間で交換可能なリクエストの情報に関する通信が必要である．システム内に同一のデータの複数コピーを持つようなシステムでは更にこれらのデータの無矛盾性——これを**相互無矛盾性** (mutual consistency) と呼んでいる——を保証することが必要となる．

4.2 ロック制御方式と OS の関係

同時実行制御を実際にインプリメントする場合にはシステムのオペレーティングシステム (OS) が提供するプロセスや同期の取り方などに関するアプローチが大きく関係する．

OS は上記の観点から大きく分けて**メッセージ指向****のものと**プロシージャ指向*****のものとがある[13]．それぞれの特徴，およびこれを用いたロック制御方式の特徴を以下にのべる．

(1) メッセージ指向の OS の場合 (図-4 参照)

- バックエンド対応にあるきまった数のプロセスが存在する．
- 同期制御はプロセス間でメッセージを交換することによって行われる．このためプロセス間で共用さ

---

* 例えば文献 1) の「分散型データベース技術」pp. 931-938 参照

** ここで同時に実行するとは，リクエスト $r1$ と $r2$ が同時刻にそれぞれ独自にデータベースに作用している状態を言う．したがって (例) のリクエストの場合のような矛盾が生じうる．

*** $r1$ と $r2$ を同時に実行した時に，給料を x として $x+2$, $2x$, $(x+2)*2$, $2x+2$ のものに更新されるものがでてくる．

* リクエスト $r1$ が操作対象とするデータ域 Q1 とリクエスト $r2$ が操作対象とするデータ域 Q2 に重なりが生じない（例えばクラスタが異なる）場合，2つのリクエスト $r1$ と $r2$ が共に削除，検索，挿入である組合せ以外でも交換可能となりうるが，挿入，更新時には，Q1，Q2 自体がリクエスト実行前後で変化するために，常に交換可能とはならない．

** message-oriented

*** procedure-oriented

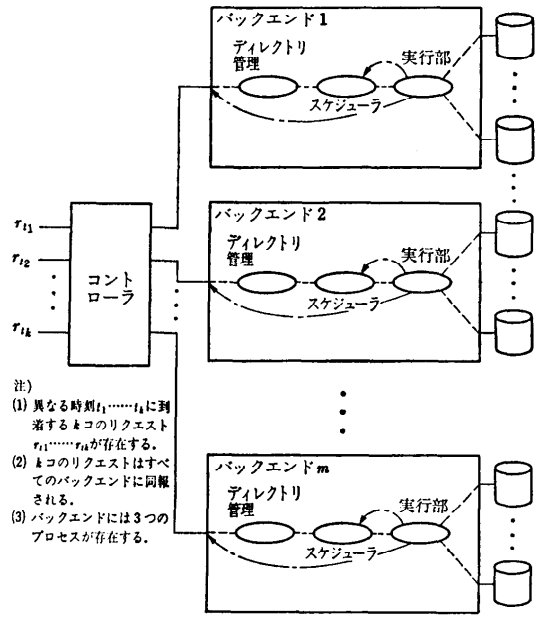


図-4　メッセージ指向の場合の同時実行制御

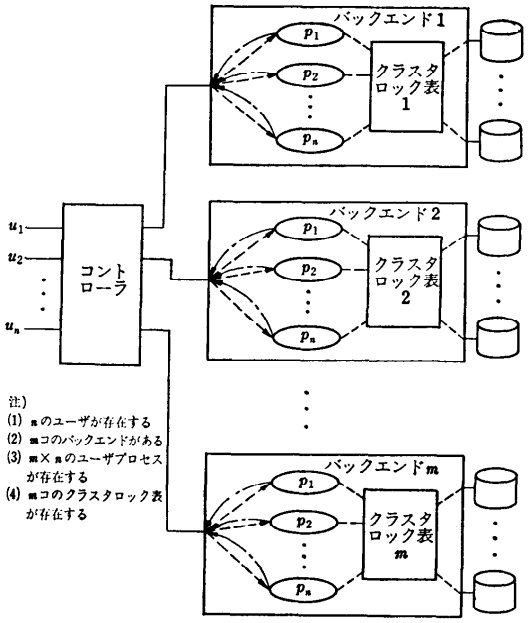


図-5　プロシージャ指向の場合の同時実行制御

れるデータは限定されている.

- プロセスはシステムの立上り時に生成され，システムの終了時に消滅する.
- ロックをかける単位はリクエスト対応となる.

ロック制御に必要なテーブルは各々のバックエンドで作成され管理される.

(2) プロシージャ指向の OS の場合 (図-5 参照)

- ユーザ対応に可変個のプロセスが存在する.
- 同期制御は，主記憶の共有領域に存在するロックデータによって制御される.
- プロセスの生成・消滅ははげしく行われる.
- ロックをかける単位は，トランザクション(一連のリクエストからなるユーザの処理要求)である.

一般に多くのデータベースシステムではプロシージャ指向のアプローチを採用しており，ロック表による同時実行制御を行っているが，Stonebraker[13] はメッセージ指向のアプローチの方が効率がよいと主張している．またメッセージ指向のアプローチでは，プロセス相互の通信量が少なく同期をとるためのオーバヘッドが小さく，リクエスト単位によってキメの細かい同期制御ができるなどの利点もある．そこで以下では，この方式を採用した同時実行制御の実現例について述べる.

4.3 同時実行制御の実現例

属性モデルを用いてクラスタ化によって分散化を行った MDBS では，リクエストのタイプ (検索，更新，挿入，削除) ごとに**使用見込** (to-be-used) と**使用中** (being-used) の2つのロックモードを設定してロック制御を行い，先に述べた交換可能および互換可能なリクエストを利用して並列性を最大限に利用しようとするものである．このシステムではデータはいずれか1つのバックエンドしか存在しないため相互無矛盾性を考える必要はない．また挿入リクエストはただ一つのバックエンドでのみ実行されるので交換・互換可能性については 表-4 のごとく比較的簡単に決められる.

これらのリクエストはクラスタを単位としてアクセスするため，各バックエンドでクラスタ対応に待

表-4　リクエスト相互の関係 (CPN 表)

	削除 (D)	挿入 (I)	更新 (U)	検索 (R)
削除 (D)	C	P	N	N
挿入 (I)	P	C	P	P
更新 (U)	N	P	N	N
検索 (R)	N	P	N	C

C：互換可能 (Compatible)
P：交換可能 (Permutable, though incompatible)
N：Not Compatible and not Permutable

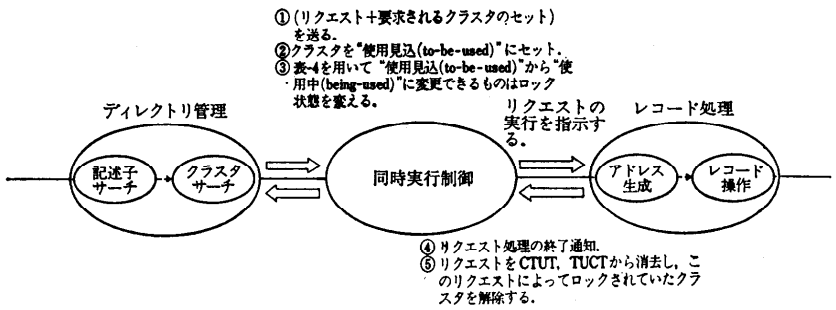


図-6　同時実行制御の手順

表-5　クラスタ－トラヒックユニット対応表(CTUT)の例

クラスタ番号	トラヒックユニット (TU)		
C1	TU1 I BU	TU2 I BU	TU3 U BU
C2	TU4 I TBU	TU3 U BU	
C3	TU4 I BU	TU5, R1 D BU	
C4	TU5, R2 U TBU	TU6 I BU	

TU1とTU2は互換可能で実行中. TU3はロックを"使用中"に変更したが, UとIは互換可能ではないので, TU3は待たされている.

TU3とTU4は交換された.

TU5, R1は, これがTU5というトランザクションの一部であることを除けば, TU4と交換可能.

TU: Traffic Unit　TBU: To Be Used
R: Request　BU: Being Used
D: Delete
I: Insert
U: Update
R: Retrieve

表-6　トラヒックユニット－クラスタ対応表(TUCT)の例

トラヒックユニット	リクエスト		
TU1 I (1リクエスト)	C1 BU		実行中 ←
TU2 I (1リクエスト)	C1 BU		実行中 ← (互換可能)
TU3 U (1リクエスト)	C1 BU	C2 TBU	C1待ち ←
TU4 I (1リクエスト)	C2 BU	C3 BU	実行中 ← (交換可能)
TU5 R1D, R2U (2リクエスト)	C3 BU	C4 TBU	C3待ち
TU6 (1リクエスト)	C4 BU		C4待ち

ち行列を作って管理する.

さてメッセージ指向の場合には**4.2**で述べたように各バックエンドの中で同時実行制御を含むいくつかのプロセスが生成される. **図-6** にこのプロセス間で行われる同時実行制御の手順を示す.

ここで用いられるロック制御表（プロシージャ指向の場合のロック表）は **表-5** に示すように, クラスタとトラヒックユニットの対応表として実現される. (この場合は5コの単一リクエストと2つのリクエストから成るトラヒックユニット (TU) について示してある.)

リクエストがバックエンドに受付けられるとそのリクエストは処理に必要なクラスタに**使用見込**のロックをかける. このリクエストが実行可能となるためには必要なクラスタをすべて**使用中**のロックに変える必要がある. こうして必要なクラスタがすべて**使用中**になったリクエストは実行に移されるが, 表-5には一つのリクエストを処理するのに必要なクラスタに関する情報がない. このためこの情報をトラヒックユニットとクラスタの対応表として作成し使用する*(**表-6**).

表-5, 表-6, 図-6によって, 同時実行制御の手順, 互換・交換可能なリクエストの関係, 実行中・待ちのリクエストの状態が理解できると思う. このような表は各バックエンドで作成されて管理される.

5. むすび

複数バックエンドデータベースシステムに適していると考えられる属性モデルの概念, 利点, ハンドリングの例などについて解説した.

現在までに多くのデータモデルが提案されていることから考えても, すべてのデータベースシステム, すべてのアプリケーションに適合するデータモデルというものはない. それぞれのデータモデルには, それに適合したデータモデル, データベース管理システム, システム構成がある. ここで述べた属性モデルデータ

* 表-6は表-5の逆表になっている.

ベースシステムも万能という訳ではなく，分割分散型のデータベースに有効な方式として紹介した．

属性モデルデータベースシステムはデータをレコードやクラスタといった比較的小さな単位で扱える反面，各種テーブルの維持・管理，ハンドリングなどに問題点を持っていると考えられる．MDBS システムは現在 DEC 社 VAX-11/780 を制御プロセッサ，pdp-11/45 をバックエンドとして用いて開発試作が進められており，評価結果の報告も順次なされていくことになっている．この設計・試作の結果は今後のデータベースシステムの構築，とりわけ分散型データベースシステムの構築にとって大変参考になることが期待できる．

参考文献

1) 大特集：データベース技術，情報処理，Vol. 23, No. 10 (1982).
2) Hsiao, D. K. et al. : Data Base Computers, Advances in Computers, Vol. 19, Academic Press. Inc. (1980).
3) Hsiao, D. K. et al. : Design and Analysis of a Multi-Backend Database System for Performance Improvement, Functionality Expansion and Capacity Growth (part I, II), The Ohio State Univ. Tech. Rep. OSU-CISRC-TR-81-7 & 8 (1981).
4) Kerr, D. S. et al. : The Implementation of a Multi-Backend Database System (MDBS) : Part I-Software Engineering Strategies and Efforts Towards a Prototype MDBS, The Ohio State Univ. Tech. Rep. OSU-CISCR-TR-82-1 (1982).
5) Higashida, M. et al. : The Implementation of a Multi-Backend Database System (MDBS) : Part II-The First Prototype MDBS and the Software Engineering Experience, Naval Postgraduate School Tech. Rep. NPS-52-82-008 (1982).
6) He, Xin-Gui et al. : The Implementation of a Multi-Backend Database System (MDBS) : The Design of a Prototype MDBS, Proceedings of the International Workshop on Database Machines, pp. 421-510 (1982).
7) Hsiao, D. K. et al. : A Formal System for Information Retrieval from Files, Comm. ACM, Vol. 13, No. 2, pp. 67-73 (1970).
8) Banerjee, J. et al. : Concepts and Capabilities of Database Computer, ACM trans. Database Syst., Vol. 3, No. 4, pp. 347-384 (1978).
9) 植村俊亮，前川 守：データベースマシン（情報処理叢書 1），pp. 62-74, 情報処理学会 (1980).
10) Auer, H. : RDBM——A Relational Database Machine, Tec. Rep. 8005, Univ. of Braunschweig (June 1980).
11) Dewitt, D. J. : DIRECT——A Multiprocessor Organization for Supporting Relational Data Management Systems, Proc. of the Fifth Annual Symposium on Computer Architecture (1978).
12) Banerjee, J. et al. : Database Transformation, Query Translation and Performance Analysis of a Database Computer in Supporting Hierarchical Database Management, IEEE Trans. Softw. Eng., Vol. 6, No. 5, pp. 91-109 (1980).
13) Stonebraker, M. : Operating System Support for Database Management Comm. ACM, Vol. 24, No. 7, pp. 412-418 (1981).

(昭和 58 年 5 月 12 日受付)