

空間 OLAP

・関係モデルの空間表現とその応用・

牧之内 顕文 龍 浩志*

九州大学大学院システム情報科学研究院

* 九州大学大学院システム情報科学府

OLAP には、多次元データベースと言われるキューブモデルに基づく MOLAP と、関係データベースをそのまま用いる ROLAP との 2 つがある。一般に、MOLAP はその操作が早い、アドホックな検索・分析が出来ないが、MOLAP では、アドホックな検索・分析が比較的容易であるが、その結果を得るのが遅いと言われている。また、MOLAP モデルは空間データベースに近似している。本稿では、われわれが研究している空間 OLAP についてモデル、実装法について構想を述べる。まず、関係空間という概念を提示し、その上での演算を定義する。演算は、関係モデルと OLAP の演算を含み、空間演算で拡張したものである。また、実装法についても言及する。

Spatial OLAP

・ Definitions and Operation

Akifumi Makinouchi Hiroshi Ryu

Graduate School of ISEE, Kyushu University

Spatial OLAP is proposed to replace ROLAP and/or MOLAP. In the spatialOLAP, a relation is represented by a space, called relation space, which consists of a set of points each of which has a tuple value as its measure. The points in the set is placed in a k-dimensional space whose axes represent attributes of the relation. Operators which work on the relation spaces are defined. The relation spaces are implemented by multi-dimensional indexes like R*-trees. A preliminary experiments were done and some issues to be investigated have identified.

1. 始めに

OLAP の概念[7]が提唱されて久しい。この後、商用の OLAP 製品が多数出荷されている。しかし、「関係データベース」とビジネスデータ分析ツールとを一体化して一つのソフトウェア製品とするという考え方はもっと古くからあり、例えば、日本では、PLANNER[9]がその嚆矢である。OLAP 製品には 2 種類あると言われている[3]。一つは、Cube モデルによる OLAP (多次元 OLAP 又は MOLAP と呼ばれる) 製品であり、他は関係データモデルをそのまま使うものである。前者は、ひらたく言えば、多次元配列に基づくモデルであり、後者のものより早い、しかし、アドホックな分析(アドホックな問い合わせを含む)が出来にくいと言われている。一方、後者(Relational OLAP, ROLAP と呼ばれる)は、アドホックな分

析には強いが、遅いと言われている。MOLAP では、分析は次元と呼ばれる複数の軸を基に行われることから、多次元データベースとも呼ばれ、モデル化の提案がなされている [1]。また、次元軸を数学の座標軸と見なせば、幾何空間と類似することから、多次元データベースと空間データベースとの類似性が指摘されている[1]。多次元データベースの実装では、空間データベースの Filtering に使われる多次元索引、例えば R*-tree が利用されることがある[2]。この実装技術の面からも両者の類似性が指摘され得る。我々は、以前から空間データベースの研究を進めてきた[6]。それら研究から得た知見に基づき、関係モデルに空間データタイプを導入するに試みも行ってきた[5]。本稿では、関係モデルの空間表現を定義し、上記の多次元データベースモデル[1]との異同を明らかにする。さらに、実装面での課題とその研究について述べる。最後に、本モデルの拡張について論じて本稿の結論とする。

2. 関連研究

本節では、まず多次元データベースモデル[1]に従って、MOLAP の概要を説明する。データはキューブまたはデータキューブと呼ばれる箱に詰められる。この箱はプログラミング言語に於ける多次元配列に似ている。配列の特定次元の「添字」に当たるものがキューブの特定次元軸上のメンバである。図 1 は「製品」の「月」別の「売上」を示している。「月」と「製品」とがこのキューブの次元である。「売上」は「(データ)セル」中に格納される。「0」はセルが無いことを示す。「売上」が示されているセルは当然存在する。この図には対応するセルが無いが、「セル」が存在することを示すのに「1」が使われる。軸上の座標を示すのが「メンバ」である。従って、幾何空間用語で言えば、図 1 のキューブは 2 次元空間を示し、座標("テレビ", "2 月")で指定される点(セル)には売上"20"があ(入っている)ことになる。セル中のデータは、一般にメジャー(measure)と呼ばれ、次元(のメンバ)とは区別される。一般的には、メジャーが次元となったり、その逆は許されない場合が多いが、[1]では、それを可能にしている。[1]のモデルでは、このキューブに次の 6 つの操作を定義している。

- (1)Push:セルに次元のメンバを挿入する
- (2)Pull:セル中の値の属性に対応した新たな次元を生成
- (3)Destroy Dimension:キューブの次元を削除
- (4)Join:2 つのキューブを結合
- (5)Restriction:指定された条件を満たすセルの集合を検索
- (7)Merge:次元の複数のメンバをひとまとめにする(Roll up に相当)

このモデルの特徴は「同じ位置に複数のセルの存在」を許さないことである。これは、配列のモデルを考えると自然である。しかし、このことから、上記操作に不自然な制約を課することになる(これについては、次節で言及する)。次に、多次元索引を用いた OLAP 実装法研究([2],[10])について述べる。[2]は、OLAP でよく用いられる aggregation (select... from...where... groupby...)結果をビューとして保存する。その時、1 つの R*-tree に複数の異なる結果を格納することにより、検索格納スペースを節約する試みである。[10]は、R*-tree に変わる多次元索引 UB-tree[4]を用いた Group-by 処理高速化のための整列アルゴリズムの提案である。

3. 関係モデルの空間化

「空間化」とは、関係(relation)を OLAP キューブに似た空間に写像し、その空間を操作することにより、関係データベースや OLAP と同様な機能を実現することを言う。この統合により、このモデルがうまく実装されれば、「MOLAP よりは柔軟性があり、RROLAP よりは高速操作できる」システムが実現できると期待される。さらに、「広がり」のあるデータを扱うデータベースシステムの実現の可能性もある[5](空間オブジェクトは通常、広がりがある。空間オブジェクトではないデータに広がりをを持たせる)。以下、そのための必要な定義について述べる。

3.1 関係空間

関係を空間表現する基本的考え方は

- ・ 関係の各タプルを、各属性を軸(axe)とする空間上の点で表現する
- ・ 各点は一つのタプル値(メジャー)を持つ(持たないこともある)
- ・ 各軸の座標値から定まる(空間)座標位置には複数の点が存在してよい

この考え方に従って、関係空間を次のように定義する。

(1) n 次の関係 $R(A_1, A_2, \dots, A_n)$ を k 次元関係空間 $R^k(D_{i1}, D_{i2}, \dots, D_{ik}) \langle A_{ik+1}, A_{ik+2}, \dots, A_m \rangle$ で表現する時

$(D_{i1}, D_{i2}, \dots, D_{ik})$ は空間 R^k の次元軸を、 $\langle A_{ik+1}, A_{ik+2}, \dots, A_m \rangle$ は、空間 R^k 中の点を持つエレメントの属性を表す。但し、 $\{D_{i1}, D_{i2}, \dots, D_{ik}, A_{ik+1}, A_{ik+2}, \dots, A_m\} = \{A_1, A_2, \dots, A_n\}$ 。

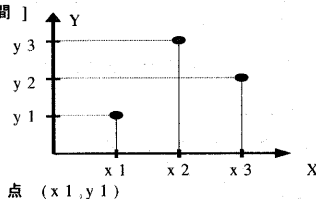
0でない値をもつデータセルを空間上の点としてみる

[キューブ]

Y			
y3	0	1	0
y2	0	0	1
y1	1	0	0
	x1	x2	x3

データセル y1 x1

[幾何空間]



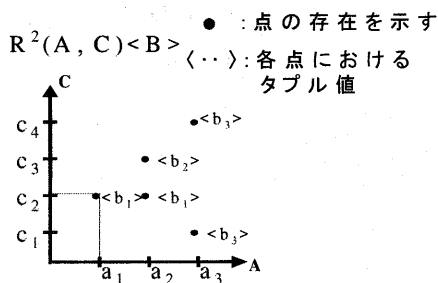
OLAPの考えと幾何空間の見え方は近い

(2) R のタプル $t_j \langle a_{j1}, a_{j2}, \dots, a_{jn} \rangle$ は k 次元空間 R^k の点 $p_{R^k}(d_{j1}, d_{j2}, \dots, d_{jk})$ に写像される。また、その点はタプル $\langle a_{jk+1}, a_{jk+2}, \dots, a_{jn} \rangle$ をエレメント(メジャー)として持つ。但し、タプル $\langle d_{j1}, d_{j2}, \dots, d_{jk}, a_{jk+1}, a_{jk+2}, \dots, a_{jn} \rangle$ は t_j のパーミュテーションの1つであり、 d_{jm}, a_{jk+1} はそれぞれ D_{im}, A_{ik+1} の属性値である。

関係 $R(A, B, C)$ の関係空間 $R^3(A, B, C) \langle \rangle$ と $R^2(A, C) \langle B \rangle$ の表現を示す。

$R(A, B, C)$

A	B	C
a ₁	b ₁	c ₂
a ₂	b ₁	c ₂
a ₂	b ₂	c ₃
a ₃	b ₃	c ₁
a ₃	b ₃	c ₄



3.2 関係空間の操作

関係空間の操作は、関係空間をオペランドとし、結果も関係空間である演算で行われる。

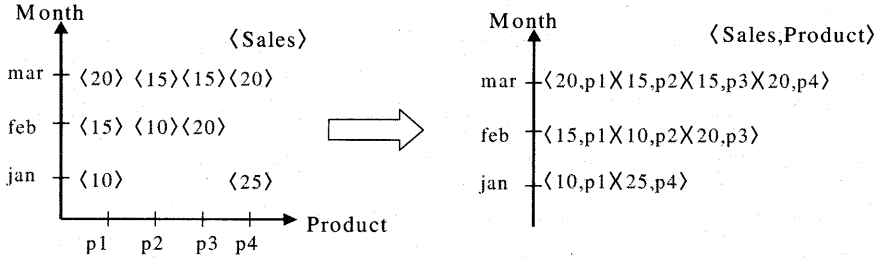
以下、代表的な演算子の定義を書く。

(1) Push: 次元上のメンバをエレメントに挿入

$$\text{Push}(R^k(D_1, D_2, \dots, D_k) \langle A_{k+1}, A_{k+2}, \dots, A_n \rangle, D_l) = S^{k+1}(D_1, D_2, \dots, D_{i-1}, D_{i+1}, \dots, D_k) \langle A_{k+1}, A_{k+2}, \dots, A_n, D_l \rangle$$

- ・ 点 $p_{R^k}(d_1, d_2, \dots, d_k) \in R^k$ は点 $p_{S^{k+1}}(d_1, d_2, \dots, d_{i-1}, d_{i+1}, \dots, d_k) \in S^{k+1}$ に写像される
- ・ 点 p_{R^k} のエレメントに値 d_i を追加したエレメントを点 $p_{S^{k+1}}$ に持たせる

例：次元 Product を Push: $\text{Push}(R^2(\text{Month}, \text{Product}) \langle \text{Sales} \rangle, \text{Product})$



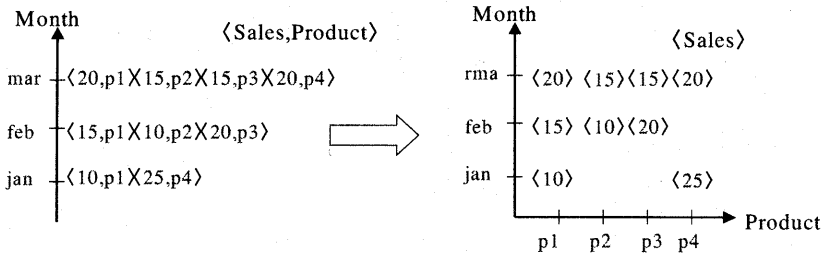
この'Push'は[1]が'Push'と異なるのは、後者では、Push される軸がそのまま残ることである。そのため、表現される情報が冗長となる。

(2) Pull: エレメントの属性を次元に変換

$$\text{Pull}(R^k(D_1, D_2, \dots, D_k) \langle A_{k+1}, A_{k+2}, \dots, A_n \rangle, A_{k+j}) = S^{k+1}(D_1, D_2, \dots, D_{i-1}, D_{i+1}, \dots, D_k, A_{k+j}) \langle A_{k+1}, A_{k+2}, \dots, A_{k+(j-1)}, A_{k+(j+1)}, \dots, A_n \rangle$$

- ・ 点 $p_{R^k}(d_1, d_2, \dots, d_k) \in R^k$ は点 $p_{S^{k+1}}(d_1, d_2, \dots, d_k, a_j) \in S^{k+1}$ に写像される
- ・ 点 p_{R^k} のエレメントに値 d_i を追加したエレメントを点 $p_{S^{k+1}}$ に持たせる

例：Element の Product を Pull: $\text{Pull}(R^1(\text{Month}) \langle \text{Sales}, \text{Product} \rangle, \text{Product})$



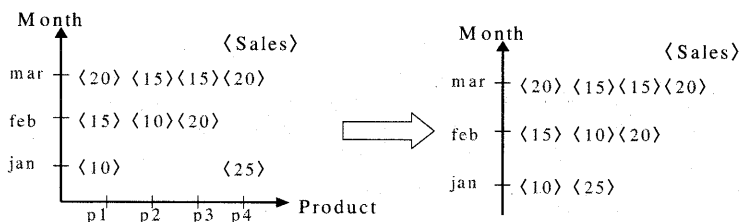
(3) Destroy Dimension: Dimension を削除

$$\text{Destroy}(R^k(D_1, D_2, \dots, D_{i-1}, D_i, D_{i+1}, \dots, D_k) \langle A_{k+1}, A_{k+2}, \dots, A_n \rangle, D_i) = S^{k+1}(D_1, D_2, \dots, D_{i-1}, D_{i+1}, \dots, D_k) \langle A_{k+1}, A_{k+2}, \dots, A_n \rangle$$

- ・ 点 $p_{R^k}(d_1, d_2, \dots, d_k) \in R^k$ は点 $p_{S^{k+1}}(d_1, d_2, \dots, d_{i-1}, d_{i+1}, \dots, d_k) \in S^{k+1}$ に写像される

- 点 p_{R^k} のエレメントと同じエレメントを点 $p_{S^{k+1}}$ に持たせる

例：次元 Product を削除：Destroy ($R^2(\text{Month}, \text{Product}) < \text{Sales} >, \text{Product}$)



[1]に於けるこの操作では、データセルの重複(一つの位置に2つのセルが存在)を許さないため、上記の例のようなことは許されない。許されるのは、消去される軸上にメンバが唯一1つある時のみである。この条件は非常に厳しい。本稿の'Destroy Dimension'は関係モデルの Projection と等価である。

(4) Cartesian: 2つの空間のデカルト積

これについては記述を省略

(5) Join: 2つの空間の結合

これについては記述を省略

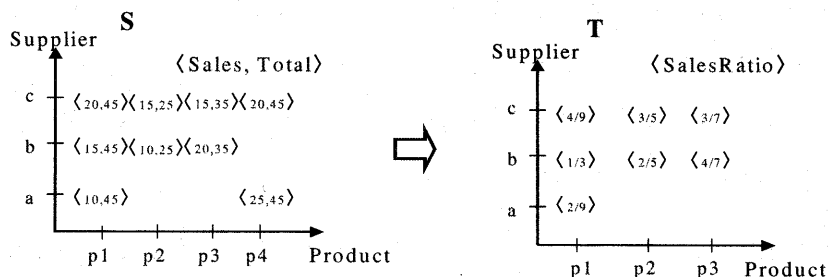
(6) Apply: エレメントへの関数の作用

$$\text{Apply} (R^k(D_1, D_2, \dots, D_k) < A_{k+1}, A_{k+2}, \dots, A_n >, \{f_1, f_2, \dots, f_n\}) = S^k(D_1, D_2, \dots, D_k) < B_{k+1}, \dots, B_m >$$

- 点 $p_{R^k}(d_1, d_2, \dots, d_k) \in R^k$ は点 $p_{S^k}(d_1, d_2, \dots, d_k) \in S^k$ に写像される
- 点 p_{S^k} のエレメントを $< b_{k+1}, b_{k+2}, \dots, b_m >$ 、点 p_{R^k} のエレメントを $< a_{k+1}, a_{k+2}, \dots, a_n >$ とすると、 $b_{k+i} = f_i(\{a_{k+1}, a_{k+2}, \dots, a_n\})$ の任意の部分集合

関係空間 S のエレメントに $f = \text{Sales} / \text{Total}$ を適用

$$\begin{aligned} \text{Apply} (S^2(\text{Product}, \text{Supplier}) < \text{Sales}, \text{Total} >, \{ \text{Sales} / \text{Total} \}) \\ = T^2(\text{Product}, \text{Supplier}) < \text{SalesRatio} > \end{aligned}$$



(7) Aggregate: 同一位置の点集合とそれらのエレメントとを一点に集約

$$\text{Aggregate} (R^k(D_1, D_2, \dots, D_k) < A_{k+1}, A_{k+2}, \dots, A_n >, \{f_1, f_2, \dots, f_n\}) = S^k(D_1, D_2, \dots, D_k) < B_{k+1}, \dots, B_m >$$

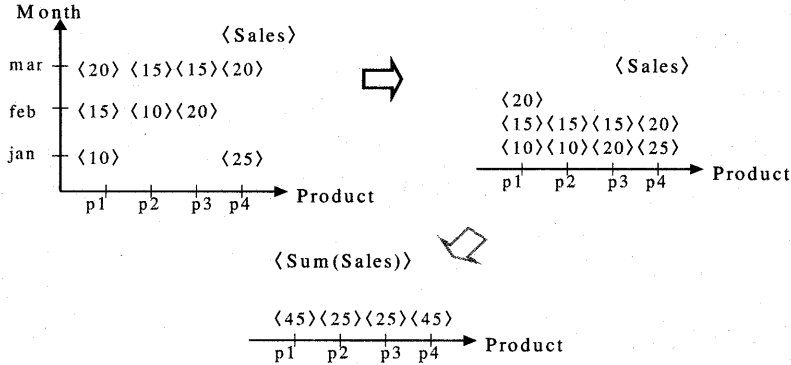
- f は同一属性の属性値マルチ集合に関する集約関数

- ・ 関係空間 R^k 中の同一位置にある p 個の点集合 $\{p_{R^k}^i(d_1, d_2, \dots, d_k)\} (i = 1, \dots, p)$ を点 $p_{S^k}(d_1, d_2, \dots, d_k) \in S^k$ に写像する.
- ・ $\langle a_{k+1}, a_{k+2}, \dots, a_n \rangle$

次元 Month を削除した後、 $f = \text{Sum}$ を用いてエレメント $\langle \text{Sales} \rangle$ をグループ化

・ $\text{Destroy}(R^2(\text{Month}, \text{Product}) \langle \text{Sales} \rangle, \text{Product}) = S^1(\text{Product}) \langle \text{Sales} \rangle$

・ $\text{Aggregate}(S^1(\text{Product}) \langle \text{Sales} \rangle, \{\text{Sum}\}) = T^1(\text{Product}) \langle \text{Sum}(\text{Sales}) \rangle$



(8) Restrict(1): 指定された空間条件を満足する関係空間を求める $\text{Restrict}(R^k(D_1, D_2, \dots, D_k) \langle A_{k+1}, A_{k+2}, \dots, A_n \rangle, SP, PC) = S^k(D_1, D_2, \dots, D_k) \langle A_{k+1}, A_{k+2}, \dots, A_n \rangle$

$A_{k+2}, \dots, A_n, SP, PC) = S^k(D_1, D_2, \dots, D_k) \langle A_{k+1}, A_{k+2}, \dots, A_n \rangle$

・ SP: Spatial Predicate, PC: Polygon Complex

・ $p_{R^k}(d_1, d_2, \dots, d_k) \in R^k$ かつ $SP(PC, p_{R^k}) = \text{true}$ であれば点 $p_{R^k}(d_1, d_2, \dots, d_k) \in S^k$

・ 両空間の対応する点が持つエレメントは同じ

(9) Restrict(2): 与えられた点から「 k 個の近傍点」を求める

$\text{NearestNeighbors}(R^k(D_1, D_2, \dots, D_k) \langle A_{k+1}, A_{k+2}, \dots, A_n \rangle, p, k) = S^k(D_1, D_2, \dots, D_k) \langle A_{k+1}, A_{k+2}, \dots, A_n \rangle$

・ p : 与えられた点 k : 求める近傍の数

・ $p_{R^k}(d_1, d_2, \dots, d_k) \in R^k$ かつ $\text{NearestNeighbors}(p, p_{R^k}) = \text{true}$ であれば点 $p_{R^k}(d_1, d_2, \dots, d_k) \in S^k$

・ 両空間の対応する点が持つエレメントは同じ

4. 関係空間の実装

関係空間を実装するのに多次元索引を使う候補としては、UB-tree と R-tree 族とが考えられる。この時、次の2つの事項が解決すべき課題として存在する。

4.1 'Group by'演算の高速化

今、関係 $R(A_1, A_2, \dots, A_n)$ に対応する関係空間 $R^k(A_1, A_2, \dots, A_k) \langle A_{k+1}, A_{k+2}, \dots, A_n \rangle$ を多次元索引で実装すると、属性 $A_1, A_2, \dots, A_k$ に索引が張られることになる。この時、属性集合 $\{A_1, A_2, \dots, A_k\}$ の部分集合で Group

byする(つまり、Aggregateをとる)時の「性能」が課題である。属性集合 $\{A_1, A_2, \dots, A_k\}$ の内、多くの属性でRestrictされた結果をAggregateする場合はともかく、1つまたは2つの属性でGroup byする時、単純な関係のmerge-sortによる実装と比べてどうなるか。

UB-treeを使った場合のアルゴリズムと実験の結果が[10]に報告されている。この場合、多次元データ空間上のデータは多次元索引によるクラスタリングの結果、整列を行う次元で部分的に整列されているという特性を利用するものである。そこで、テーブルにアクセスする際にある程度整列された順序でアクセスをおこなうことにより、リレーション全体を同時に整列する必要がなく、高価な外部ソートを避け、処理の効率化を図るというものである。提案しているアルゴリズムに関し、1次元のみの索引を用いた方法、merge-sortを用いた方法、と性能比較を行っており、いずれの方法よりも高速な処理時間を実現している。

われわれは、R*-treeを使った時の整列アルゴリズムを考え、それを実装し、簡単な予備実験を行った。UB-TreeとR*-Treeとは、空間の分割の仕方に違いがあるが、UB-Treeのアルゴリズムの処理の基本的な考え方を適用し、リレーションに対し、整列を行う次元で部分的に整列された順序でアクセスを行うアルゴリズムの実装を行った。しかし、処理領域がデータ空間に占める割合が大きくなると、単純なmerge-sortと比較して性能が非常に悪くなるという結果が出た。一般に索引を用いる場合、結果として得られる集合が限られている場合には、ディスクアクセス回数の減少により高速化を図れる。また、UB-Treeと比較して、R*-Treeではデータへアクセスするためのパスが唯一に決まらないために生じるディスクIO時間の増加や、merge-sortと比較するとディスクに対する高価なランダムアクセスを伴うなどの原因が考えられるが、更なるアルゴリズムの検討や実験が必要である。また、複数の次元で整列を行うためのアルゴリズムの検討も課題となる。

4.2 異なるタイプの次元からなる多次元索引の実装法

UB-treeでは、整数次元からなる多次元索引の実装法は知られているが、実数(float)次元のもの、あるいは、文字列次元に関しては、私の知る限りまだその実装法が報告されていない。従って、UB-treeで関係空間を実現することは、現在のところ難しい。

R-tree族は、基本的に次元は「数値」である。次元が整数次元であろうと、実数次元であろうと、数値タイプの次元であれば、時に応じて「変換」することによりR-treeやR*-treeでも実装できる。しかし、文字列次元が次元に加わると話は別である。[11]では、文字列属性のみからなるデータをR*-treeにより索引付けして検索を高速化した。しかし、すべての次元が文字列タイプでなければならないという制約がある。

関係空間の実装には、次のような条件を満たす多次元索引の実装法を考案しなければならない。

(1) 異なるタイプの次元軸からなる空間

ある軸は数値であり、別の軸は文字列である。この場合、空間上に合理的な「距離」は定義できない。

(2) 軸上の座標値集合(属性の定義域)は、「全順序」であることのみ条件とする

例えば、2つの文字列間の距離を合理的に定義するのは難しいと考える。しかし、任意の2つの文字列間の大小(順序)を想定するのは問題ない。

5. 結論

本稿で提案したモデルはまだ途中段階であり、もっと推敲する必要がある。その時に考えるべきことは2つある。一つは、広がりを持つデータをどう導入するかである。他は、関係モデルやOLAPにない操作だが、役に立つものを導入することである。候補としては、関係空間の「畳みこみ」である。

さらに、このモデルを実装する上での課題は大きい。「MOLAPより柔軟で、ROLAPより早い」システムが出来るかに本提案の存在意義がかかっている。

参考文献

- [1] Agrawal R., Gupta A., and Sarawagi S. Modeling Multidimensional Databases. Proc. ICDE, 1997 pp. 232-243
- [2] Kodidis Y. and Roussopoulos N. An alternative Storage Organization for ROLAP Aggregate Views Based on Cubetrees. SIGMOD, 1998 pp. 249-258
- [3] Chaudhuri S. and Dayal U. Data Warehousing and OLAP for Decision Support. SIGMOD Tutorial, 1997 pp. 507-508
- [4] Bayer R. The Universal B-Tree for Multidimensional Indexing. TUM-I9637, Technical University of Munich, September, 1996
- [5] Makinouchi A. and Kuroki S. Enhancing Databases by introducing Spatial Data Types for Non-Geographical Applications. In Advances in Multimedia and Databases for The New Century: A Swiss/Japanese Perspective (Ed. Masunaga Y. and Spaccapietra S., World Scientific), Kyoto Japan, Dec., 1999, pp. 99-105
- [6] 黒木 進, 牧之内顕文 位相空間データモデル Universe の空間, 時間, 時空間データ表現 情報処理学会論文誌, 第 40 巻, 第 5 号, 1999
- [7] Codd E.F., Codd S.B., and Salley C.T. Providing OLAP (on-line analytical processing) to user-analysts: An IT mandate. Technical report, E.F. Codd and Associates, 1993
- [8] Jagadish H.V., Koudas N., Srivastava D. On Effective Multi-Dimensional Indexing for Strings. ACM SIGMOD 2000, May, 2000
- [9] 牧之内顕文, 手塚正義, 神田康敬, 甲田一也 関係データベースを中核にした計画管理情報システム 情報処理学会論文誌 第 25 巻, 第 1 号, 1984 年 3 月 pp. 19-29
- [10] Volker Markl, Martin Zirkel, Rudolf Bayer Processing Operations with Restrictions in RDBMS without External Sorting: The Tetris Algorithm, ICDE 1999
- [11] H.V.Jagadish, Nick Koudas, Divesh Srivastava On Effective Multi-Dimensional Indexing for Strings ACM SIGMOD 2000