DEWS2006 3C-o2

SuperSQL 処理系における中間データキャッシュの実装と効率化

有澤　達也†　　遠山　元道††

† 慶應義塾大学 デジタルメディア・コンテンツ統合研究機構
〒 108–8345 東京都港区三田 2–15–45
†† 慶應義塾大学 理工学部 情報工学科
〒 223–8522 神奈川県横浜市港北区日吉 3–14–1
E-mail: †ari@dmc.keio.ac.jp, ††toyama@ics.keio.ac.jp

あらまし　SuperSQL 処理系では，INVOKE 関数を用いて動的にデータベースを参照し HTML 文書を生成することができるが，HTML 文書に変換する前段階の中間データをキャッシュすることで，新たに要求された質問文に対してその質問構造を比較することにより，適切なキャッシュを用いて生成することを以前提案した．本稿では，その中間データのキャッシュの実装方法について述べる．そして，それぞれのキャッシュ手法の比較評価を行う．また，SuperSQL 処理系のキャッシュの汎用性について調べることで，格納領域についてより効率的にキャッシュデータを格納する手法について述べる．

キーワード　SuperSQL, Web キャッシング, 関係データベース

An implementation and efficient method of caching intermediate result for SuperSQL system

Tatsuya ARISAWA† and Motomichi TOYAMA††

† Research Institute for Digital Media and Content, Keio University.
Mita 2–15–45, Minato-ku, Tokyo, 108–8345 Japan
†† Department of Information and Computer Science, Faculty of Science and Technology, Keio University.
Hiyoshi 3–14–1, Kouhoku-ku, Yokohama-shi, Kanagawa, 223–8522 Japan
E-mail: †ari@dmc.keio.ac.jp, ††toyama@ics.keio.ac.jp

Abstract　SuperSQL system can generate a result of dynamic evaluation by relational database using INVOKE function. Before we proposed the methods of caching the intermediate result for INVOKE function. And we proposed using the suitable cache for similar request by comparing the data structure of SuperSQL query. In this paper, we describe how to implement the caching systems for intermediate result and evaluate the caching systems. Then we consider a generalness of these cache systems and describe about efficiently storing cash data.

Key words　SuperSQL, Web Caching, Relational Database

1. はじめに

現在，データベースと連係して Web ページを提供するシステムが多く存在する．例えば，ユーザから Web アクセスがあった際に，あらかじめ決められた項目の情報を格納しておいたデータベースへ問い合わせ，その結果をもとにサーバ側で動的に HTML を生成し提供するようなシステムである．このようなシステムでは，動的に変化する部分を分離することで，統一した HTML のページデザインでの提供が可能になる．

このようなシステムを実現するために，慶應義塾大学遠山研究室で開発している SuperSQL [1], [2] では，HTML の生成に関して INVOKE 関数が用意されている．この INVOKE 関数を用いると，該当する箇所に Java Servlet を用いて実装された SuperSQL 処理サーバへのリンクを生成することができる．ユーザがリンクをクリックするたびに，INVOKE 関数で指定された条件をパラメータとして渡し，動的に HTML 文書を生成してユーザに提示を行う．

Web サーバ上に Servlet で実装された SuperSQL 処理系では，通常リクエストを受け取る度にデータベースへの問い合わせを行い，その結果を整形して出力する．そして, [5] では生成結果の HTML 文書をキャッシュすることにより，同一の質問文に対して迅速に生成結果を返すことができるようになった．ま

た, [4] では，HTML 文書に変換する前段階の中間結果をキャッシュする手法を提案した．これは，動的に処理する SuperSQL 質問文の構造を，キャッシュ生成時の質問文の構造と比較することにより，取得項目が等しいなどの類似の質問文に対して，データベースへのアクセスを行わずに HTML 文書を生成する手法である．

本稿では, [4] で示した中間結果をキャッシュする手法に関して，キャッシュ処理の実装を行うことにより，各キャッシュ手法の有効性について評価を行う．また，キャッシュデータの汎用性を考慮することにより，効率のよいキャッシュデータの構築方法について述べる．

2. SuperSQL 処理系におけるキャッシュ

2.1 SuperSQL と INVOKE 関数

SuperSQL は，データベースへの問い合わせと同時に，その検索結果の構造化を行い，指定された対象メディアへの出力を行う処理系である．SuperSQL では特に SQL のターゲットリストを拡張した構文 TFE を用い，データのレイアウトを規定する．結合子は属性等を「,」（横）,「!」（縦）,「%」（深さ）,「#」（時間）で区切ることによって，それぞれの方向にレイアウトすることを意味する．また，反復子は，反復させたい部分を大括弧「[ ]」で囲み，その直後に反復方向を結合子と同じ記号で記述することで，大括弧の外側にある属性をキーとしてグルーピングすることができる．

Servlet 版の SuperSQL 処理系では，INVOKE 関数を用いることで，複数の SuperSQL 質問文の間にリンクによるナビゲーション機能を実現することができる．INVOKE 関数は，呼び出すべき質問文のファイル名や属性値をパラメータに埋め込んだリンクを生成し，ユーザはこのリンクをたどることで Web サーバ上の Servlet 版の SuperSQL 処理系を動的に呼び出して HTML を生成することが可能である．INVOKE 関数の書式は以下の通りである．(注1)

INVOKE(*att*, filename=*query_file*,
condition=*selection_condition*)

INVOKE 関数の引数には，リンクされる文字列を表す属性 *att* の他，リンクで呼び出した際に評価する SuperSQL 質問文のファイル名を示す *query_file* と，その SuperSQL 質問文に付加する検索条件に用いるための条件文 *selection_condition* を指定する．*selection_condition* では，ダブルクオーテーション内に書かれた文字列と，データベース属性名を組合せることで，*query_file* で指定された SuperSQL 質問文に条件を加えることができる．HTML を生成する際に，指定されたデータベース属性はタプルごとにデータベースから取得された値に置き換えられる．

この INVOKE 関数は SuperSQL 処理系によって，属性 *att* を評価した文字列 *value* に図 1 のような Servlet 版 SuperSQL へのリンクを生成する．このリンクには，INVOKE で指定された情報や Web サーバの設定により，表 1 のパラメータが含まれる．

(注1)：SuperSQL の TFE における関数の記述方法が [4], [5] から変更されている．

value

図 1 INVOKE 関数から生成されるリンク
Fig. 1 A hyperlink corresponding to INVOKE function

表 1 INVOKE 関数から生成されるリンクのパラメータ
Table 1 Parameters of the hyperlink corresponding INVOKE function

パラメータ名	説明
dbname	接続データベース名
query_file	利用する SuperSQL 質問文
condition	*query_file* に付加する検索条件

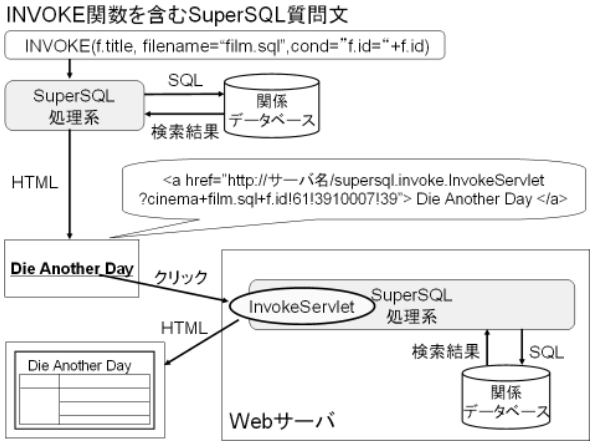

図 2 INVOKE 関数を用いた動的な HTML 文書の生成
Fig. 2 Process of dynamically generating a HTML document using INVOKE function

INVOKE 関数を用いた動的な HTML 文書を生成する処理の流れについて図 2 に示す．リンク経由で呼び出された Servlet 版 SuperSQL 処理系の InvokeServlet クラスでは，まず最初に，*query_file* で指定されたファイルの SuperSQL 質問文に *condition* で指定された条件を付加した SuperSQL 質問文を生成する．そして，この SuperSQL 質問文に対してデータベースに問い合わせを行い生成された結果をユーザに提示する．

このように Servlet 版 SuperSQL 処理系では，ユーザから INVOKE 関数によって生成されたリンクをたどるたびに，SuperSQL 処理系からデータベースへのアクセスが発生する．従来の研究では，SuperSQL 処理系で生成されたオブジェクトをキャッシュとして保存することにより，データベースへの繰り返しのアクセスを避けることを提案した．次節では，このキャッシュの手法について述べる．

2.2 キャッシュ手法

SuperSQL 処理系では，与えられた SuperSQL 質問文に対して，図 3 の 3 つの操作を経てデータベースの検索結果から HTML 文書を生成している．処理される SuperSQL 質問文が

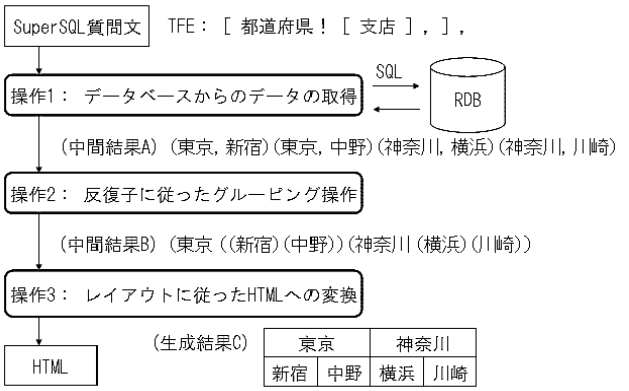

図 3　SuperSQL 処理系における HTML 文書の生成過程

Fig. 3　Process of generating HTML documents on SuperSQL system

複雑な場合は，データベースへのアクセスやデータの構造化等にかかる時間が長くなってしまう．そこで以前の研究では，これら 3 つの操作後に生じる中間結果をキャッシュすることで，データベースへアクセスせずに迅速に HTML を生成する手法を提案した．

[5] では，必要とする SuperSQL 質問文内で結合されている表のタプルが更新されていないならば，同一の SuperSQL で生成される HTML 文書は同一になることに着目し，SuperSQL 処理系で生成された HTML 文書を Web サーバにキャッシュする手法を提案した．この手法を用いることで，一度 SuperSQL 処理系で生成したことのある SuperSQL 質問文が再び与えられた場合，SuperSQL 処理系を利用せずにキャッシュに蓄えられている結果からユーザに迅速に HTML 文書を提示することが可能になった．

また, [4] では，SuperSQL 処理系で生成された中間結果を Web サーバにキャッシュする手法を提案した．キャッシュの対象とするのは，図 3 における操作 1 の後の中間結果 (A) の「DB スナップショット」，操作 2 の後の中間結果 (B) の「木構造データ」である．この手法を利用すれば，一度 SuperSQL 処理系で生成したことのある同一ではないが類似の SuperSQL 質問文に対して，キャッシュに蓄えられている結果からデータベースへのアクセスを行わない方法で，HTML 文書を提示することが可能である．

この [4] で提案した中間結果のキャッシュ手法について，本稿では実装を行い評価を行う．以下では，この DB スナップショットのキャッシュと木構造データのキャッシュについて概要を示す．

2.2.1　DB スナップショットのキャッシュ

DB スナップショットのキャッシュは，SuperSQL 処理系がデータベースに問い合わせた結果を，そのままの形で Web サーバに保存する方法である．

例えば，映画館とその上映タイトルという関連をあらわす表があり，この表に対して次のような二つの SuperSQL 質問文があるとする．

```
(a) GENERATE html [ 映画館 , [ タイトル ] ! ] !
    FROM 映画館データ
(b) GENERATE html [ タイトル , [ 映画館 ] ! ] !
    FROM 映画館データ
```

(a) は「映画館ごとに上映している映画を見たい」という質問文であり，(b) は「映画のタイトルごとに上映映画館を見たい」という質問文である．

この二つの SuperSQL 質問文に対して，SuperSQL 処理系では，図 3 の操作 1 にあたるデータベースへの問い合わせとして，それぞれ以下に示す SQL が発行される．

```
(a') SELECT 映画館，タイトル FROM 映画館データ
(b') SELECT タイトル，映画館 FROM 映画館データ
```

この (a') と (b') の SQL は，属性の射影の順序が異なるだけであり，この SQL の結果生じるビュー，すなわち図 3 の中間結果 A は全く同等の情報を持っている．言い替えれば，(a') の結果から (b) の HTML 文書を生成することが可能であり，(b') の結果から (a) の HTML 文書を生成することが可能である．

そこで，(a) による動的な HTML 文書生成が行われる過程で，図 3 の中間結果 A にあたる，(映画館，タイトル) を属性にもつビューを DB スナップショットとしてキャッシュしておくとする．後に，(b) によって HTML 文書を動的に生成する場合，SQL の結果が変わらないことが保証されていれば，先のキャッシュを利用して，データベースへのアクセスなしに HTML を生成することが可能である．

この例のように，対象となるデータベースのビューが同じ情報になる SuperSQL 質問文が複数ある場合に，この DB スナップショットをキャッシュとして格納することで，以後の生成に対してデータベースへのアクセスを省略することが可能である．ここで蓄積するキャッシュを DB スナップショットのキャッシュと呼ぶ．

2.2.2　木構造データのキャッシュ

木構造データのキャッシュは，DB スナップショットのデータに対して SuperSQL 質問文内の反復子で指定されたグルーピングを行った後の木構造をもったデータを，キャッシュとして Web サーバに保存する方法である．

再び 2.2.1 節の映画館データベースの例を用いる．この表に対して次のような二つの SuperSQL 質問文があるとする．

```
(a) GENERATE html [ 映画館 , [ タイトル ]! ]!
    FROM 映画館データ
(c) GENERATE html [ 映画館 ! [ タイトル ]! ],
    FROM 映画館データ
```

(a) は 2.2.1 節でも用いたもので，(c) は「映画館ごとに，その下に上映タイトルを縦に並べ，これを映画館があるだけ横方向に反復する」という質問文である．この二つの質問文は，データをレイアウトする方向だけが異なり，HTML を構成するために必要なデータは全く同じである．したがって，DB スナップショットのキャッシュを利用することができる．

しかし，2.2.1 節の関係とは異なり，質問文 (a) と (c) ではどちらも「映画館ごとに上映タイトルをグルーピングする」操作が続くので，中間結果 A が等しい場合，操作 2 によって生成された木構造データの中間結果 B は二つの質問文で等しい．

したがって，(a) を生成する時に生成される木構造データの中間結果 B から (c) の HTML 文書を生成することが可能であり，その逆も可能である．

そこで，(a) による動的な HTML 文書生成が行われる過程で，図 3 の中間結果 B にあたる，映画館ごとに上映タイトルをグルーピングしたデータを木構造データとしてキャッシュしておくとする．後に (c) によって HTML 文書を動的に生成する場合，SQL の結果が変わらないことが保証されていれば，先の木構造データを利用することにより，データベースへのアクセスを行わず，またグルーピング操作をしないことでより迅速に HTML を生成することが可能である．

この例のように，対象をなるデータベースのビューが等しく，属性間のグルーピングの関係が等しい SuperSQL 質問文が複数ある場合に，木構造データをキャッシュとして格納することで，以後の生成に対してデータベースへのアクセスおよびデータのグルーピング操作を省略することが可能となる．ここで蓄積するキャッシュを木構造データのキャッシュと呼ぶ．

3. DB スナップショットのキャッシュの構築方針

DB スナップショットのキャッシュを利用可能にするには，キャッシュする中間結果だけではなく，そのデータを生成した SQL の情報をメタデータとして格納し比較に用いることが必要となる．[4] では DB スナップショットのキャッシュを構築する際に必要なメタデータに関して，SuperSQL 質問文から発行された SQL に対して以下の情報を格納することが必要であると述べている．

- SELECT 節に書かれている属性集合のリスト
- FROM 節に書かれている表名リスト
- WHERE 節以下の条件

しかしながら，DB スナップショットのキャッシュではキャッシュの対象はデータベースへの問い合わせから得られる出力であることから，キャッシュとして実装する際には，SELECT 節に記述されている属性はその射影の順序に依存せずにキャッシュ対象となるかどうかを判断する必要がある．そこで，属性をリストではなく集合で格納する方が効率よく比較を行うことできる．一方，属性を集合としてメタデータを記録した場合，キャッシュを取得した際の属性の順序がわからなくなってしまう．

そこで，利用している属性に関しては，キャッシュが利用可能かどうかの比較のための属性の集合として記録と，キャッシュを再構築するための順序を保存した属性リストの二つをメタデータとして記録する．

後に SuperSQL 質問文でキャッシュを利用する際には，まず質問文から属性集合を抽出し，キャッシュにあるメタデータの属性集合と比較することにより，利用可能なキャッシュが存在するかどうかの判定を行う．ここで存在すると判定された場合，該当する DB スナップショットのキャッシュを取得する．その後，SuperSQL 質問文から生成された属性リストと，キャッシュに付与されている属性リストを比較しながら，それぞれのタプルの順序を必要とされている順序に変更する．

このようにして，属性の指定の順序が異なる場合でも，DB スナップショットのキャッシュを適用することができる．

また，SuperSQL 質問文に利用している属性については，FROM 節で記載されている表名に別名が指定される場合や表名が省略される場合がある．この場合には，FROM 節を確認し，別名が指定されているものは実際の表の名前に置き換え，省略された表名については補完したうえでキャッシュのメタデータとして記録する．

WHERE 節以下の条件については，同じ条件を異なる方法で記述可能であることを考慮する必要がある．しかし，比較のコストが大きくなることと，Web サイト構築者が SuperSQL 質問文における条件の表現方法を統一することができることから，本稿の実装では複雑な解析は行わないこととする．そこで WHERE 節以下の条件については，表名を補完もしくは置換したものをそのままメタデータとして記録する．

4. 木構造データのキャッシュの構築方針

木構造データのキャッシュを利用可能にするには，グルーピング操作を行った後の木構造データの中間結果だけではなく，そのデータを生成した SQL の情報およびグルーピング操作についてメタデータとして格納し，比較に用いることが必要となる．[4] では木構造データのキャッシュを構築する際に必要なメタデータに関して，SuperSQL 質問文から発行された SQL とそのグルーピング操作に対して以下の情報を格納することが必要であると述べている．

- SELECT 節に書かれている属性集合のリスト
- FROM 節に書かれている表名リスト
- WHERE 節以下の条件
- グルーピング木 [6]

木構造データのキャッシュでは，グルーピングの関係が等しいことがキャッシュが適用可能な条件の一つである．グルーピング木は，どの属性がどの属性によってグルーピングされているかを，木構造の親子関係を用いて表したものである．

そこでグルーピング木の格納方法は，再帰的にグルーピングしている属性集合の部分を一つの集合としてまとめたものをメタデータとする．このように集合として格納することにより，キャッシュが利用可能かどうかを判定することが容易になるが，一方，抽出されたキャッシュの属性およびグルーピングの順序がわからなくなるため，SuperSQL 質問文で指定された出力の順序に並べることができない．そこで，グルーピング木のメタデータに関しても，DB スナップショットの属性集合と同様に，リストとして順序を保存したグルーピング木を同時に記録することで，抽出されたキャッシュを指定された順序に再構成することを可能にした．

また，木構造データのキャッシュを利用するためには，DB スナップショットのキャッシュについて利用可能な条件は全て満たす必要がある．しかし，グルーピング木が等しいことが確認できれば，SQL で指定された属性集合が等しいことを意味しているため，SQL で指定する属性集合のメタデータは必要なく，WHERE 節以下の部分についてのみのメタデータを記録する．

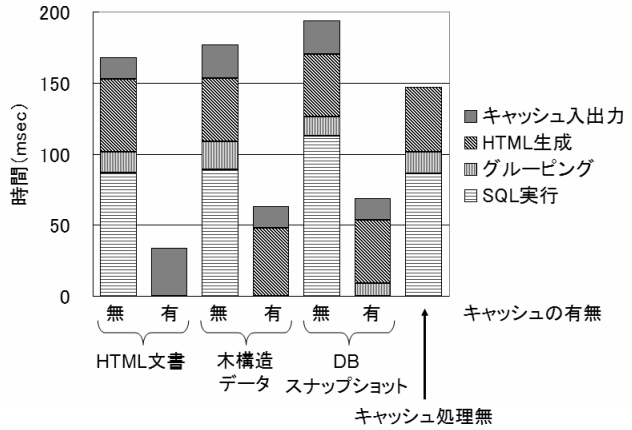

図 4　キャッシュ処理による HTML 文書生成時間
Fig. 4　Time of generating HTML documents using SuperSQL cache system

5. 実装評価

3., 4. 節での方針に従い，SuperSQL 処理系に DB スナップショットのキャッシュおよび木構造データのキャッシュを行うシステムを実装した．実装言語には JAVA を用い，キャッシュのメタデータの構築には，Apache Tomcat [7] 上の Servlet を用いて永続化されたインスタンスを共有する形で実装した．実験に用いたマシンは CPU Pentium M 1.6GHz，メモリ 1.5GB，OS は Windows XP Professional である．

まず，中間データのキャッシュの有効性を確かめるために，今回実装した DB スナップショットのキャッシュおよび木構造データのキャッシュを用いた場合に加え，HTML 文書のキャッシュを用いた場合および全くキャッシュ処理を行わない場合についても同条件で実験を行い，質問文から HTML を生成する処理時間について比較を行った．処理時間は以下に分けて取得した．

- データベースへの問い合わせ時間
- フラットデータの木構造化を行う時間
- レイアウトに従って HTML を生成する時間
- キャッシュの読み書きにかかる時間

この実験により，2 度目以降の再取得時間がキャッシュ処理を行わない場合と比較して，どの程度減少させることができるかを確認する．実験の結果は図 4 のようになった．

キャッシュ処理を導入することで，要求された SuperSQL 質問文のキャッシュが既に存在する時には，キャッシュ処理を行わない場合と比較して，いずれのキャッシュ処理を利用した場合も処理時間を大幅に短縮できることがわかる．特に，相対的に処理時間の長いデータベースへのアクセスを省略できることは，ユーザへの応答時間を短縮するだけでなく，SuperSQL システムやデータベースへの負荷を軽くすることができるため，有効である．

中間結果のキャッシュを利用した時と HTML キャッシュを利用した時を比較すると，中間結果のキャッシュを利用した場合，2 倍程度の時間がかかっている．この時間は，主に中間結果から HTML を生成するための時間である．しかし，中間結果のキャッシュは，別のレイアウトや別媒体への出力にも利用可能であるという点において，HTML 文書のキャッシュより優位であるといえる．

表 2　保存されたキャッシュのファイルサイズ
Table 2　Size of cached object

使用したキャッシュの種類	ファイルサイズ (Kbyte)
DB スナップショット	19
木構造データ	14
HTML 文書	64
生成された HTML 文書	64

逆に，要求された SuperSQL 質問文のキャッシュがまだ存在しない質問文の場合は，キャッシュ処理を行わない場合と比較してキャッシュを保存するための処理が必要なために，時間が多くかかっている．しかし，このキャッシュを導入したことによる時間に関するオーバーヘッドは，キャッシュの蓄積時およびキャッシュの利用時共に，全体の処理時間と比べて 1 割程度と小さいことがわかる．したがって，キャッシュを繰り返し利用するような環境下では，キャッシュを使った処理を行うことが有効であると考えられる．

次に，保存されたキャッシュのファイルサイズについて測定したところ，表 2 の結果となった．表 2 から，キャッシュとして格納されるファイルサイズについて，二種類の中間結果のキャッシュのサイズは，HTML 文書のキャッシュと比べて 3 分の 1 から 4 分の 1 程度に縮小していることがわかる．そのうえ，HTML 文書のキャッシュは単一の質問文に対してのみ利用できるのに対して，二種類の中間結果のキャッシュについてはより広範囲でキャッシュを利用できるため，効率よくディスクにキャッシュを構築できることがわかる．

これら中間データの二種類のキャッシュについて，それぞれ領域的な効果が大きいことがわかったが，両方のキャッシュを維持することはコストが大きい．そこで，限られたキャッシュ領域で運用するために，DB スナップショットと木構造データのキャッシュを効率的に格納していくことについて，次節で考察する．

6. キャッシュの効率化

DB スナップショットのキャッシュは最も汎用的にキャッシュに適用できるが，SuperSQL 処理系から発行される SQL 文が JOIN を多く含むなど複雑になった場合，データベースからの結果が非常に大きくなることがあり，多くのディスク領域が必要となることが予想される．キャッシュに利用できるディスクは限られているので，DB スナップショットのキャッシュはキャッシュ管理が難しい．

そこで，DB スナップショットのキャッシュを木構造データのキャッシュで代用することについて考察する．木構造データは，グルーピング操作まで終わっているため，実験結果からもわかるように，中間結果の大きさを DB スナップショットと比較して小さくできるからである．しかし，木構造データのキャッシュ

タイトル	映画館
THE 有頂天ホテル	渋東シネタワー
THE 有頂天ホテル	新宿文化シネマ
輪廻	渋東シネタワー
輪廻	新宿コマ東宝
…	…

図 5　SQL 文の問い合わせ結果
Fig. 5　A result of the query

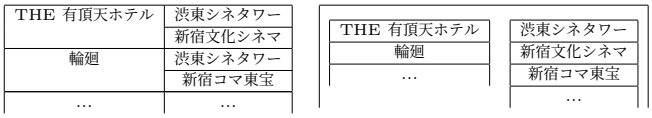

図 6　(I)(II) に対応する木構造データ
Fig. 6　Nested data corresponding to SuperSQL query (I) and (II)

は，グルーピング操作が行われているので，DB スナップショットの時点では保持している属性間の関連の情報が必ずしも保存されるとは限らない．このことについて例示し，考察する．

6.1　木構造データにおける属性間の関連の喪失

例えば，映画データベースへ問い合わせる次の二つの SuperSQL 質問文を考える．

```
(I)  GENERATE html [タイトル,[映画館]!]!
     FROM 映画館データ
(II) GENERATE html [タイトル]!,[映画館]!
     FROM 映画館データ
```

(I) は上映されているタイトルごとに映画館のリストを並べる質問文であり，(II) は上映されているタイトルのリストと映画館のリストを並べる質問文である．

これら二つの SuperSQL 質問文から SuperSQL 処理系で必要となる SQL 文は，いずれも

```
SELECT タイトル, 映画館 FROM 映画館データ
```

となり，その結果として例えば図 5 のタプル集合を得る．このタプル集合に対して SuperSQL 質問文に従いグルーピングを行った結果，それぞれ図 6 のような木構造データを得ることができる．(注2)

これら (i) と (ii) の木構造データから元の SQL の結果である図 5 が生成できるかを確認するために，木構造データのグルーピングされている属性値を組み合わせる．(i) については，グルーピングのキー項目である「タイトル」に対してグルーピングされている全ての「映画館」を組み合わせることで，元の SQL の結果である図 5 が復元できる．しかし，(ii) については，「タイトル」のリストと「映画館」のリストの間の関連が失われているため，元の SQL の結果を復元できない．

したがって SuperSQL 質問文には，木構造データから元の SQL の結果，つまり DB スナップショットと同等のものが復元できるものと復元できないものが存在する．そこで，復元できる SuperSQL 質問文であることがわかれば，木構造データのキャッシュのみで，それを生成したサイズの大きい DB スナップショットのキャッシュを代替できることになる．そこで次節では，木構造データのキャッシュから DB スナップショットのキャッシュを復元できる条件について考察する．

6.2　DB スナップショットに復元可能な木構造データ

木構造データのキャッシュを DB スナップショットのキャッシュに復元できる場合があることを前節で述べた．この例から復元が可能な SuperSQL 質問文の特徴について考える．

前節の例で復元できる SuperSQL 質問文 (I) と復元できない (II) では，データベースにアクセスする SQL 文が等しいものであり，違いはグルーピングの適用方法の違いだけである．復元できない (II) では同一階層に複数のグルーピングを含んでおり，そのグルーピング間の関連が失われている．

ここで，グルーピング操作の逆の操作，つまり木構造データから SQL からの取得結果へ戻す操作を平坦化と定義する．すると SuperSQL におけるグルーピング操作が平坦化が可能であるための十分条件について，以下のとおり定理として示される．

［定理 1］ SuperSQL におけるグルーピング操作は，同一階層に複数のグルーピング操作が存在しなければ，木構造データは平坦化可能である．

［証明 1］ 同一階層についてグルーピング操作が一つしかないので，グルーピングを含まないキー項目となる属性集合を A，グルーピングされる属性集合を B とし，グルーピングを行う前の表を $R = \{A, B\}$ とする．

この場合，グルーピング操作は，キー項目である属性集合 A の値が等しいものごとにグルーピングされる属性集合 B を集める操作である．属性集合 A の特定の属性値の集合 a_k に集められたタプルは，グルーピング操作後に属性集合 A の特定の属性値集合 a_k に対して，それぞれのタプルの属性集合 B の射影を取った部分のタプルを持つ集合 $B(a_k)$ として保存される．ここで，属性集合 A の部分にはグルーピング操作が存在しないため，属性値の集合 a_k については単一の値集合のみしかとることができない．

平坦化は，キー項目である属性値の集合 a_k の値とグルーピングされた集合 $B(a_k)$ に含まれるタプルの全ての組合せをとる操作である．属性値の集合 a_k は単一の属性値の組しかとらないことから，射影をとったタプルは全て元のタプルを構成することが可能である．

また，平坦化後に元のタプルに存在しないタプル $t' = \{a', b'\}$ があったと仮定する．平坦化後にタプル t' が存在するためには，グルーピング操作をしたときに，キー項目が a' である属性値集合 $B(a')$ に射影 b' が存在する．キー項目が a' であるから，この射影前のタプルは $\{a', b'\}$ となり，これはタプル t' そのものであり，仮定と矛盾する．

したがって，グルーピング操作が同一階層に一つしか存在しない場合，グルーピング操作を行ったタプルを平坦化したときに，グルーピング操作前にあったタプルは全て復元可能であり，グルーピング操作前に存在しなかったタプルは存在しない．つまり平坦化可能であるといえる．　□

この証明から，複数のグルーピングが同一階層に存在しないことが，木構造データのキャッシュから DB スナップショット

(注2)：わかりやすくするため，図 6 ではリスト形式でなく木構造の形で表現する．

を復元できる十分条件といえる．したがって，複数のグルーピングが同一階層に存在しない場合は，DB スナップショットをキャッシュとして持つ代わりに，木構造データをキャッシュとして格納することで代用することが可能である．DB スナップショットの要求時には，対応する木構造データのキャッシュについて平坦化を行った結果を提供すればよい．

本節で提案した効率化の手法については，今後実装の予定である．

7. おわりに

本稿では，SuperSQL 処理系における動的な HTML 文書の生成において，DB スナップショットおよび木構造データをキャッシュする手法について，実装を行い，キャッシュの有効性について評価を行った．

この結果，DB スナップショットのキャッシュおよび木構造データのキャッシュが，SuperSQL 処理系の動的な HTML に対して，生成時間を短縮することを示した．

また，SuperSQL 質問文において同一階層に複数のグルーピング部分が存在しない場合，DB スナップショットのキャッシュを木構造データのキャッシュで代用することができることを示した．DB スナップショットと木構造データのキャッシュを共通化することで，効率的なキャッシュ運用ができると考えられる．今後実装を行うことにより，キャッシュの共通化による効果とオーバーヘッドについて考察を行う予定である．

また，本稿で提案したキャッシュシステムを運用させるためには，キャッシュされたオブジェクトの的確な管理が必要である．キャッシュ元のデータベースが更新された際にキャッシュを破棄したり，キャッシュ領域が足りなくなった際のキャッシュオブジェクトのパージ処理が必要である．これらのキャッシュ管理についても，今後検討を行う予定である．

文　献

[1] Motomichi Toyama: SuperSQL: An Extended SQL for Database Publishing and Presentation, *Preceedings of ACM SIGMOD '98 International Conference on Management of Data*, pp. 584 – 586 (1998).

[2] SuperSQL HOME PAGE,
http://www.db.ics.keio.ac.jp/ssql/index.html

[3] C. Agrawal, J. Wolf, P. Yu: Caching on the World Wide Web, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 11, No.1, (1999).

[4] 有澤達也, 石川恭子, 遠山元道: SuperSQL の INVOKE 処理における中間データのキャッシュ,
日本データベース学会 Letters, Vol.3, No.1, pp. 33 – 36, (2004).

[5] 有澤達也, 石川恭子, 遠山元道: SuperSQL 処理系における INVOKE 関数に対するキャッシュ機構,
情報処理学会研究報告, DBS-131-037, (2003).

[6] 有澤達也, 遠山元道: SuperSQL 処理系におけるグルーピング操作の効率的な実装,
データ工学ワークショップ (DEWS2001), (2001).

[7] Apache Tomcat
http://tomcat.apache.org/