DEWS2004 4-C-03

SuperSQL の INVOKE 処理における中間データのキャッシュ

有澤　達也[†,††]　　石川　恭子[†]　　遠山　元道[†††]

† 慶應義塾大学大学院 理工学研究科 開放環境科学専攻　〒 223–8522 神奈川県横浜市港北区日吉 3–14–1
†† 慶應義塾大学 ITC 本部　〒 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,kyoko}@db.ics.keio.ac.jp, ††toyama@ics.keio.ac.jp

あらまし　SuperSQL 処理系の INVOKE 関数を用いると，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．この際に，生成結果の HTML 文書をキャッシュして格納するといった研究を以前行い，同一質問に対する生成コストを軽減することができるようになった．しかしながら，パーソナライゼーション等で同一出力データに対してレイアウトだけ異なった結果が必要な場合，キャッシュを利用できなかった．そこで本研究では，HTML 文書に変換する前段階のデータをキャッシュし，要求された質問文の構造を比較することにより，適切なキャッシュを用いて類似質問文に対する生成コストを軽減することを提案する．

キーワード　SuperSQL, Web キャッシング, Servlet, 関係データベース

Caching intermediate result for INVOKE function of SuperSQL system

Tatsuya ARISAWA[†,††], Kyoko ISHIKAWA[†], and Motomichi TOYAMA[†††]

† School for Open and Envionmental Systems, Graduate School of Science and Technology, Keio University.
Hiyoshi 3–14–1, Kouhoku-ku, Yokohama-shi, Kanagawa, 223–8522 Japan
†† Keio University Information Technology Center.　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,kyoko}@db.ics.keio.ac.jp, ††toyama@ics.keio.ac.jp

Abstract　SuperSQL system is used for generating various HTML documents from relational database. It generates dynamic evaluation result to a database by using links generated from INVOKE function. Before we proposed caching system for SuperSQL which caches HTML documents generated dynamically by SuperSQL INVOKE function. The system has decreased the cost of regeneration for the same request later. However, cash have not been applied for generating a document which has different layout for personalization from the same data. In this paper, we propose the method of cashing the immediate data before changing into HTML documents for SuperSQL INVOKE function, and use the suitable cache for similar request by comparing the data structure of SuperSQL query.

Key words　SuperSQL, Web Caching, Servlet, Relational Database

1. はじめに

現在，Web での情報発信の手段として，ユーザからの要求時にバックエンドにあるデータベースから動的に情報を取得し，その結果から Web ページを作成して提供することがある．例えば，飲食店の情報を同一のデザインを用いて表示する際に，店の情報のみをデータベースに格納し，実際に参照される際に動的に生成して提示する等である．

慶應義塾大学遠山研究室で開発している SuperSQL [1], [2] では，このような動的に Web ページを生成するための INVOKE 関数が用意されている．INVOKE 関数で作成されたリンクには，リンク先で評価すべき SuperSQL 質問文および検索条件が埋め込まれており，リンクをクリックするたびに，CGI を用いて SuperSQL 処理系にパラメータを渡し，動的に HTML 文書を生成してユーザに提示を行う．

[5] では，この INVOKE 関数によって生成された HTML 文書をキャッシュすることで，同一リンクに対する複数回のアクセスに対して，同一の HTML 文書を迅速に返すことができるようになった．しかし，これが利用できるのは，呼び出された SuperSQL 質問文が同一であり，かつ同一の検索条件だけであ

り，例えばブラウザの大きさによってレイアウトを変えるような ACTIVIEW [4] などでは，これらのキャッシュを利用できなかった．

そこで本稿では，HTML 文書に変換する前段階のデータをキャッシュすることによって，INVOKE 関数で指示される SuperSQL 質問文の構造を，キャッシュ生成時と比較することにより，適切なキャッシュを用いて類似質問文に対する生成コストを軽減し，ユーザへの応答時間を短縮することを目的とする．

2. SuperSQL と INVOKE 関数

2.1 SuperSQL

SuperSQL は，関係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．この SuperSQL の中心となるものは，関係データベースへの問い合わせ言語である SQL のターゲットリストを拡張したもので，TFE と呼ばれる．

この TFE では，通常の SQL のターゲットリストにでてくるデータベース属性に対して，結合子によってレイアウト方向を指定したり，反復子によるデータのグルーピング，そして「@{}」により装飾情報を付加することができる．

結合子は属性等をカンマの代わりに，「,」（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，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．

特に，HTML 文書をターゲットとする場合には，結合子の「%」はリンクによる結合を表し，前の属性の文字列に後に続く TFE に対するリンクを生成する．

例えば，

`title % [ actor ]!`

といった TFE の場合は，`title` の文字列にリンクが生成され，そのリンク先にその `title` をキーとしてグルーピングされた `actor` の一覧の表を生成する．

2.2 INVOKE 関数

SuperSQL では，TFE 内に INVOKE 関数を記述することで，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，パラメータ内に呼び出すべき質問文のファイル名や属性値を埋め込んだリンクを生成し，このリンクをたどることで，SuperSQL を呼び出し動的に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．

att `%` `INVOKE(`*query_file*`,` *selection_condition*`,` *selection_attribute*`)`

INVOKE 関数内の引数については，動的に呼び出す 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* と，その条件となる値を決定するためのデータベース属性名 *selection_attribute* を指定する．

また，INVOKE 関数は属性からのリンクとして生成されるため，関数の前に「%」を用いてリンクされる文字列等を表す属性（*att*）が必要である．

この INVOKE 関数は SuperSQL 処理系によって，「%」の前に与えられた属性 *att* を評価した文字列 *value* に SuperSQL の CGI への図 1 のようなリンクを生成する．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 のパラメータが与えられている．

`<a href="http://Web サーバ名/ssql.cgi?`*query_file*
+query_path+condition+dbname`">` *value* `</a>`

図 1　INVOKE 関数から生成されるリンク

表 1　INVOKE 関数から生成されるリンクのパラメータ

パラメータ名	説明
query_file	利用する SuperSQL 質問文
query_path	*query_file* が存在するディレクトリ
condition	INVOKE 関数に付加する検索条件
dbname	接続データベース名

リンク経由で呼び出された `ssql.cgi` では，*query_path* と *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 質問文を生成し，Web サーバ内の SuperSQL 処理系に問い合わせを行う．この結果をユーザに提示する．

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．

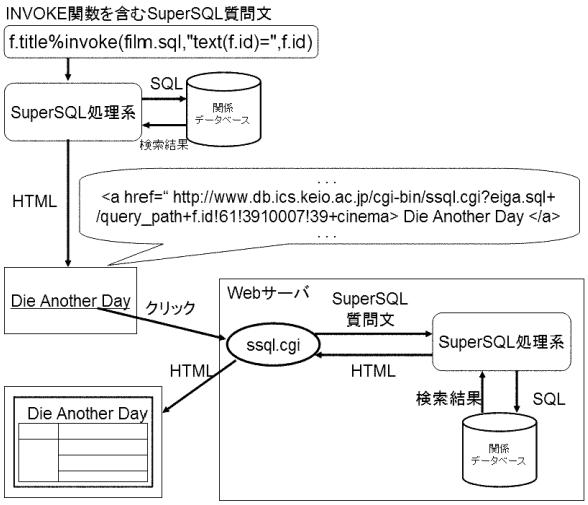

図 2　INVOKE 関数を用いた動的な HTML 文書の生成

2.3 生成結果のキャッシング

INVOKE 関数によるリンクで実行される CGI プログラム `ssql.cgi` は，要求の度に SuperSQL 処理系を呼び出している．処理される SuperSQL 質問文が複雑な場合は，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，HTML

文書を出力するまでの応答時間がかかってしまう．特にアクセスが集中すると，データベースから検索結果を取得するまでに遅延が生じてしまう．

そこで，従来の研究 [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，INVOKE 関数から作られたリンクによって指定されている CGI プログラムで，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．

2.4 キャッシュ対象の拡張

SuperSQL 処理系では図 3 のように，

- 操作 1: SQL を用いて関係データベースからフラットなデータを取得
- 操作 2: 反復子に従ってグルーピング操作を行い木構造データに変換
- 操作 3: SuperSQL 質問文のレイアウトに従って HTML 文書に変換

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．

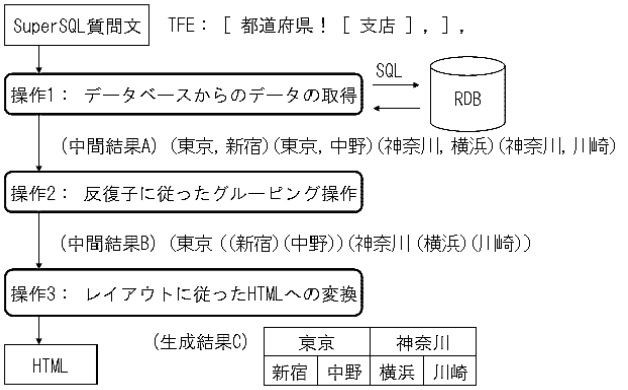

図 3 SuperSQL 質問文が与えられてから HTML 文書が生成されるまでの過程

このうち，操作 1 ではデータベースへのアクセスが必要なため，発行される SQL によっては多くの時間を要する．特に SuperSQL 質問文では，生成結果のグルーピングを生かすことのできる 1 対多や多対多の関連の結合を利用することが多いことから，データベースへの問い合わせコストが大きくなってしまう可能性がある．

また操作 2 では，グルーピングを行うために，フラットなデータに対して部分的ソートを行う必要がある．ソート操作の一部は，SQL でアクセスする際に ORDER BY 節を適切に与えること [6] で代替が可能であるが，複数のグルーピングが並列している場合では，ORDER BY 節のみではグルーピングに必要なソートを全て行うことはできない．また，操作 2 を行うことで，フラットな表からの繰り返しを含むような中間結果を，グルーピングに従って小さくすることが可能である．

以前の研究においては，操作 3 の結果の HTML 文書 (図 3 の (C) にあたる) のみをキャッシュしていた．しかし，レイアウトだけが異なる SuperSQL 質問文の場合は，操作 1,2 までは共通であるにもかかわらず，キャッシュを利用することができなかった．例えば，図 3 の例で，支店を並べる方向を縦方向にする場合，必要となるデータは一緒であるにもかかわらず，最終的な HTML 文書が異なるため，従来の HTML 文書のキャッシュ方式ではキャッシュを利用できず，SuperSQL 処理系を呼び出しデータベースへのアクセスから行う必要があった．

そこで，本稿では操作 1 の後の中間結果 (A) を「DB スナップショット」，操作 2 の後の中間結果 (B) を「木構造データ」と呼ぶこととする．そして，一度生成されたこれらの中間結果を Web サーバ側でキャッシュすることで，同一ではないが類似の SuperSQL 質問文に対して DB スナップショットや木構造データのキャッシュから HTML 文書を生成することによって，INVOKE 関数が設定するリンクによる HTML 文書の動的生成にかかる応答時間を短縮することを目指す．

本稿では DB スナップショットおよび木構造データのキャッシュについて，それぞれのキャッシュの性質について述べ，実装に必要なメタデータの設計について述べていく．

3. 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 処理系が発行した SQL をデータベース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．以下では，DB スナップショットのキャッシュの適用例を示し，DB スナップショットをキャッシュすることの有効性およびキャッシュ構築の上で必要なメタデータについて述べる．

3.1 キャッシュの適用例

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．それぞれの生成結果のレイアウトを図 4 に示す．

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 にあたる関係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
```

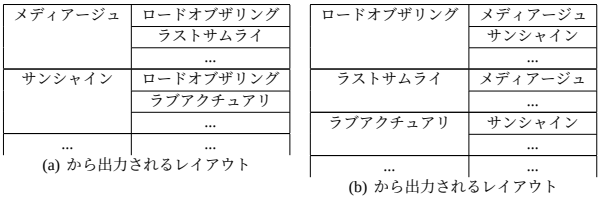

図 4　質問文 (a),(b) から生成される HTML 文書のレイアウト

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ということである．

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ことにする．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，先にキャッシュした (a) の DB スナップショットを利用して，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，対象をなるビューが等しくなるような 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，2 回目以降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．

3.2　キャッシュの有効性

DB スナップショットのキャッシュでは，グルーピング操作を行う前のデータを保持しているため，グルーピング操作のキーが変更になった場合でも利用することが可能になる．つまり，キャッシュの適用範囲は広いといえる．特に 3.1 節の例のように一つのビューに対してさまざまな面から見る場合には，このキャッシュは有効である．

また，データベースへ問い合わせる SQL が複数の表を結合するなど計算量が大きくなる場合は，データベースへの接続を省略できるため，大きな効果が期待できる．

しかしながら，DB スナップショットはビューをそのまま表形式で格納する必要があるため，非常に大きな格納領域が必要になる可能性がある．操作 2 のグルーピング操作によって，反復されるグルーピングのキー項目を一つにまとめていて，キャッシュの時点ではその恩恵を得ることができない．

3.3　キャッシュのメタデータ

DB スナップショットでは，SuperSQL 質問文から発行された SQL が同等であり，その SQL が参照する表が更新されていないことが，キャッシュを利用するための条件となる．

したがって，DB スナップショットでは，中間結果とともに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．具体的には，SuperSQL 質問文から発行された SQL に対して，

- 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

をメタデータとして保持する必要がある．

また，データベースアクセスの最適化として，一つの SuperSQL 質問文に対して複数の SQL が発行されるときがあるが，この場合はそれぞれの SQL 文に対してキャッシュを格納し利用することになる．

4.　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 スナップショットに対して SuperSQL 質問文内の反復子によって指定されたグルーピングを行ったあと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．以下では，木構造データのキャッシュの適用例を示し，木構造データをキャッシュすることの有効性およびキャッシュ構築の上で必要なメタデータについて述べる．

4.1　キャッシュの適用例

再び 3.1 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 3.1 節でも用いたもので,「映画館ごとに，その右側に上映タイトルを縦に並べ，これを映画館の分だけ縦方向に反復する」という質問文である．これに対して (c) は「映画館ごとに，その下に上映タイトルを縦に並べ，これを映画館の分だけ横方向に反復する」という質問文である．(c) の生成結果の例を図 5 に示すが，(a) の生成結果の例である図 4 と比較してもわかる通り，これらこの二つの質問文は，データの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．

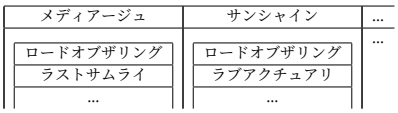

図 5　質問文 (c) から生成される HTML 文書のレイアウト

実際に，(c) の質問文に対して SuperSQL 処理系は次の (c') の SQL によってデータベースへの問い合わせを行う．

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

これは (a') と全く同じ質問文である．したがって，この時点で DB スナップショットのキャッシュが利用できることがわかる．

しかし，3.1 節の (a) と (b) の質問文の関係とは異なり，(a) と (c) の HTML 文書の生成の過程を比較すると，DB スナップショットを生成した後，その次の段階の木構造データの生成まで同一操作を行っている．言い替えると，データベースからのデータを取得後には，図 3 の操作 2 に示すように，反復子に

従ったグルーピング操作を行うが，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」といった操作が必要になり，中間結果 A が等しい状況下では，操作 2 によって生成された木構造データである中間結果 B は等しくなるということである．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，逆に，(c) を生成する時に生成される木構造データから (a) の HTML 文書を生成することが可能であるということである．

ここで，この操作 2 にあたるグルーピング操作では，再帰的にグルーピングのキーとなる項目でのソートが必要である．そのため，対象となるタプルが多くなったり，再帰的にグルーピングを行っている場合では，グルーピングに非常に多くの時間を必要としている．

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．この木構造データをキャッシュすることにより，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，先にキャッシュした (a) の木構造データを利用して，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高速に HTML を生成することが可能となる．

この例のように，対象をなるビューが等しく，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，2 回目以降の HTML 文書の生成の要求に対して，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．

4.2 キャッシュの有効性

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，中間結果の大きさが DB スナップショットと比較して小さくできる．また，最終結果となる HTML 文書と比較しても，HTML のタグでの装飾がない分だけ小さな領域しか用いないで済む．

さらに木構造データは，結合子の結合方向や，反復子の反復方向，そして装飾指定などのレイアウトに全く依存しないため，HTML 文書でキャッシュする場合より多くの SuperSQL 質問文がこのキャッシュを利用できる．

しかし，グルーピング操作によって，DB スナップショットの時点では保持している SQL の結果のタプルとしての属性値間の関連の情報が，一部欠落してしまう．したがって，同一属性集合にもかかわらず別の項目をグルーピングのキーとする SuperSQL 質問文に対しては，DB スナップショットとは異なり，この木構造データからは HTML 文書を生成できない．

4.3 キャッシュのメタデータ

木構造データのキャッシュを利用するには，DB スナップショットのキャッシュの利用条件に加えて，属性間のグルーピングの関係がキャッシュ取得時と同等である必要がある．

属性間のグルーピングの関係を表す方法として，グルーピング木 [6] がある．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．このグルーピング木が等しければ，属性間のグルーピングが等しいことを意味し，木構造データのキャッシュが適用可能である．

例えば，これまでに例に挙げてきた，SuperSQL 質問文 (a),(b),(c) に対するグルーピング木は図 6 のようになる．図

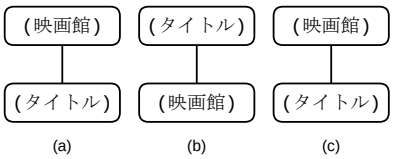

図 6　SuperSQL 質問文 (a),(b),(c) に対するグルーピング木

6 から，(a) と (c) のグルーピング木は同じであり，(b) のグルーピング木はそれらとは異なるため，(a) と (c) は木構造データのキャッシュを相互に利用できるが，(b) とは木構造データのキャッシュの内容が異なるため，利用できないと判断できる．

したがって，キャッシュのメタデータにグルーピング木を同時に格納し，新たに INVOKE 関数によって SuperSQL 質問文が与えられたときには，その SuperSQL 質問文に対するグルーピング木と比較することで，キャッシュを適用できるか否かを判断する．したがって，具体的にはキャッシュのメタデータとして，

- 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

を保持することになる．

5. キャッシュの性能比較

これまで述べてきた DB スナップショット，木構造データのキャッシュと，従来の HTML 文書のキャッシュの性能の比較をまとめると，表 2 のようになる．

表 2　3 種類のキャッシュの性能比較

	DB スナップショット	木構造データ	HTML 文書
キャッシュの汎用度	○	△	×
キャッシュ一つあたりのデータの大きさ	××	○	△
キャッシュからの HTML 文書生成時間	×	△	○

本稿の提案である，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は，従来の HTML 文書のキャッシュに比べて，同一キャッシュをより広い問い合わせに利用することが可能である半面，キャッシュからの HTML 文書を再構成する必要があり，生成までに時間がかかってしまう．

また，キャッシュ一つあたりのデータの大きさについては，HTML 文書ではタグ等で装飾されているため，グルーピングを行った直後の木構造データが最も小さくなる．グルーピングを行っていない DB スナップショットのキャッシュは，一つあたりの大きさは非常に大きいものになってしまう．

このように，DB スナップショットと木構造データ，HTML 文書のキャッシュ手法は，それぞれに一長一短の特性があるため，実際に処理される問い合わせ状況を考慮したキャッシュの設計が必要だと考えられる．

6. 実装・評価

現在，JAVA Servlet [7] を用いて DB スナップショットおよび木構造データのキャッシュの実装を行っている．実装の方針として，SuperSQL 処理系に質問文を投入した場合に，以下に示す手順で HTML 文書の生成を行う．

まず，既に格納しているキャッシュのメタデータベースから検索を行い，該当するキャッシュがあるかを確認する．

もしキャッシュが存在しなかった場合は，今まで通り操作 1 にあたるデータベースからのデータの取得から行い，HTML 文書を生成する．この過程で，SuperSQL 質問文のメタ情報をメタデータベースに格納し，操作 1 の終了後の中間結果を DB スナップショット，操作 2 の終了後の中間結果を木構造データとして，それぞれファイルの形でキャッシュを行う．

それに対してメタデータベースからキャッシュが存在することが判明した場合は，メタデータの指示するファイルを取得する．そして，そのキャッシュが DB スナップショットであれば操作 2 にあたるグルーピング操作から，木構造データであれば操作 3 にあたるレイアウトに従った HTML への変換から処理を再開することで，HTML 文書の生成を行う．

評価実験として，これらのキャッシュを用いることで，2 度目以降の再取得時間がキャッシュ無しの時と比較して，どの程度減少させることができるかを確認する．また，キャッシュの保存コストに関する評価を行い，限られたキャッシュ領域に対するキャッシュの取捨選択に関する考察を行う予定である．

7. おわりに

本稿では，SuperSQL 処理系における INVOKE 関数による動的な 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述べた．これにより，従来の HTML 文書のキャッシュだけでは利用できなかった類似の SuperSQL 質問文に対しても，キャッシュを利用することが可能になり，動的な HTML 文書生成において Web サーバの応答時間を短縮することができると考えられる．

DB スナップショットと木構造データ，HTML 文書のキャッシュ手法は，それぞれ，キャッシュの汎用性，大きさ，キャッシュからの HTML 文書生成時間に一長一短の特性を持つ．このため，実際に処理される問い合わせ状況を考慮したキャッシュの設計を行う必要があると考えられる．

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] Yoko Maeda, Motomichi Toyama: ACTIVIEW: Adaptive data presentation using SuperSQL, VLDB 2001, pp. 695 – 696, (2001).

[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] The Java Apache Project, http://java.apache.org/