SuperSQLにおけるクエリの分割・合成プリミティブおよびフォーム対応の導入

小林　武彦†　　遠山　元道††

† †† 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: †tk@db.ics.keio.ac.jp, ††toyama@ics.keio.ac.jp

あらまし　従来の SuperSQL では 1 ページに表したい内容は一つのクエリとして表さなければならず、クエリが長くなる、複雑なレイアウトにしづらい、などの問題点が存在した。そこで本論文では SuperSQL クエリを分割し、簡単なクエリの集約としてより大きなクエリを作成することにより、より多くの情報を簡単に 1 ページに載せる方法を提案する。それによりクエリを再利用することも容易になる。また以前の SuperSQL によって出力された HTML ファイルにはスタイルシートを適用することができなかった。そこで本論文では SuperSQL によって出力された HTML ファイルにスタイルシートを適用できるよう機能を拡張した。さらにフォームを使うことでインターネット上から SuperSQL 処理系を起動し、動的にファイルを生成する方法を提案する。

キーワード　SuperSQL、WEB、DB 言語

Form Interface and, Segmentation/Synthesis Primitive of SuperSQL Query

Takehiko KOBAYASHI† and Motomichi TOYAMA††

† ††Department of Information and Computer Science, Faculty of Science and Technology,
Keio University
Hiyoshi3–14–1, Kouhoku-ku, Yokohama-shi, Kanagawa, 223–8522 Japan
E-mail: †tk@db.ics.keio.ac.jp, ††toyama@ics.keio.ac.jp

Abstract　Established SuperSQL have to write one query for one page. Because of that, queries get very long and complicated. So, in this paper, we suggest the way to segmentalizate SuperSQL queries to make queries more easily on each page. As suggested way, we also suggest re-use of queries. And also, a page that created by established SuperSQL can not user CSS. Because of that, a page that created by established SuperSQL is not superior page. So, in this paper, we suggest the way to adapt CSS to a page that created by SuperSQL. In addition, by using form from Internet, we suggest the way to activate SuperSQL processor to make page dynamically.

Key words　SuperSQL, DB Language, WEB

1. はじめに

近年 1 ページあたりに表示する情報量はの多い Web ページデザインをよく目にする。特に HTML でフレームタグが推奨されなくなってからはテーブルタグやスタイルシートを駆使し、メニューバーやナビゲーションバーなどを作る必要があり、よりいっそう情報量は増加している。さらに HTML ページはスタイルシートを用いることで非常に高品質なページを作成することが可能になっている。SuperSQL はクエリにレイアウト・装飾指定を行い、質問出力を直接 HTML/PDF 等に変換する SQL の拡張言語である。しかし従来の SuperSQL ではスタイルシートを適用することができず、高品質な HTML ページの生成には極めて冗長なクエリを記述する必要があった。また 1 ページに表示したい情報は 1 つのクエリとして表現する必要があり、ページの構成が複雑な場合にはクエリは非常に長く複雑なものになってしまう。そのためメニューバーなどを含んだページを作成することは困難であった。さらに SuperSQL では HTML ファイルを作成することができるがフォームから条件を受け取って HTML ファイルを作成するためにはサーバーサイドのプログラミングを必要とした。

本論文では SuperSQL によって出力される HTML ファイルにスタイルシートを適用できるよう、拡張すると同時にそれ以外の様々なスタイルを利用できるように拡張する。さらに新たな関数として embed 関数を導入し、SuperSQL クエリを細分

化し、小さくかつ単純なクエリを組み合わせることで一つの大きなクエリを作成できるようにする。さらにフォームからSuperSQL 処理系を起動できるようにすることで「検索形アプリケーション」をプログラミングせずに簡易的に作成することを可能にする。

以下、本稿の構成を示す。まず 2. 章で SuperSQL の概要について述べる。次に 3. 章で SuperSQL による HTML 出力にスタイルシートを適応する方法について述べる。さらに 4. 章で embed 関数の実装について述べる。そして 5. 章でフォームから SuperSQL 処理系を起動させるサーブレットの実装について述べ、続く 6. 章でアンケートによる結果を述べ、7. 章で結論およびまとめを述べる。

2. SuperSQL

この章では本論文で改善を試みる SuperSQL について簡単に述べる。SuperSQL は関係データベースの出力結果を構造化し、多様なレイアウト表現を可能とする SQL の拡張言語であり、慶應義塾大学遠山研究室で開発されている [1] [2]。そのクエリは SQL の SELECT 句を GENERATE< $media$ >< TFE > の構文を持つ GENERATE 句で置き換えたものである。ここで < $media$ > は出力媒体を示し，HTML，PDF などの指定ができる．また < TFE > はターゲットリストの拡張である Target Form Expression を表し，結合子，反復子などのレイアウト指定演算子を持つ一種の式である。

2.1 結 合 子

結合子はデータベースから得られたデータをどの方向 (次元) に結合するかを指定する演算子であり, 以下の 3 種類がある。括弧内はクエリ中の演算子を示している。

- 水平結合子 (,)

データを横に結合して出力。

例：Name, Tel

name	tel

- 垂直結合子 (!)

データを縦に結合して出力。

例：Name! Tel

name
tel

- 深度結合子 (%)

データを 3 次元方法へ結合。出力が HTML ならばリンクとなる。

例：Name % Tel

name	→	tel

2.2 反 復 子

反復子は指定する方向に、データベースの値があるだけ繰りかえして表示する。また反復子はただ構造を指定するだけでなく、そのネストの関係によって属性間の関連を指定できる。例えば

[科目名]! , [学籍番号]! , [評点]!

とした場合には各属性間に関連はなく、単に各々の一覧が表示されるだけである。一方，ネストを利用して

[科目名 ! [学籍番号 , 評点]!]!

とした場合には、その科目毎に学籍番号と評点の一覧が表示されるといったように、属性間の関連が指定される。以下、その種類について述べる。

- 水平反復子 ([ ],)

データインスタンスがある限り、その属性のデータを横に繰り返し表示する。

例：[Name],

name1	name2	…	name10

- 垂直反復子 ([ ]!)

データインスタンスがある限り、その属性のデータを縦に繰り返し表示する。

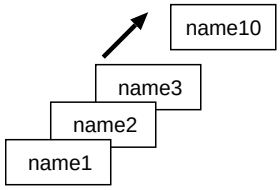
例：[Name]!

name1
name2
…
name10

- 深度反復子 ([ ]%)

データインスタンスがある限り、その属性のデータを奥行き方向 (HTML ではリンク、PDF ではページ変換) に繰り返し表示する。

例：[Name]%



2.3 装 飾 子

SuperSQL では関係データベースより抽出された情報に、文字サイズ、文字スタイル、横幅、文字色、背景、高さ、位置などの情報を付加できる。これらは装飾演算子 (@) によって指定する。

```
<属性名>@{ <装飾指定> }
```

装飾指定は” 装飾子の名称 = その内容” として指定する。複数指定するときは各々を”,” で区切る。

2.4 関 数

SuperSQL ではいくつかの関数が用意されている。ここでは代表的な関数を 2 つ紹介する。

2.4.1 imagefile 関数

imagefile 関数を用いると画像を表示することが可能となる。引数には属性名、画像ファイルの存在するディレクトリにパスを指定する。

imagefile(id, path=”./pic”)

2.4.2 sinvoke 関数 (出力メディアが HTML の場合のみ)

sinvoke 関数は FOREACH 句と同時に用いる。これらを用いることで深度結合子と同様にリンクを生成することができる。

3. HTML ページの高品質化

従来の SuperSQL では生成された HTML ページはあまり高品質とは呼べなかった。それは多種多様なスタイルの指定ができなかったことが一因である。そこで本論文では様々なスタイルの指定をできるよう SuperSQL の改良を行った。基本的に装飾子の種類を増やすことで全て対応している。今回、テーブルの表題の指定を可能にするためにテーブルの装飾子に title、テーブルの位置の指定のために tablealign を追加し、リンク先のウィンドウの指定のために sinvoke の属性の装飾子に target を追加、さらにボーダーを完全に消すためにテーブルの装飾子に tableborder を追加した。

さらに Web ページ（HTML) にはスタイルシート（CSS) という機能が存在する。スタイルシートによって Web ページは劇的に美しく、親和性、認知性の高いページにすることができる。

しかしながら 従来の SuperSQL はスタイルシートには対応していなかった。そのため生成される Web ページはデザイン性、親和性、認知性などに乏しいものとなってしまう場合があった。この理由として「ワンソースマルチユース」という考えがある。スタイルシートを導入してしまうとそのクエリは PDF 生成には利用しづらくなってしまう。しかしこのメリットを多少失ってでもスタイルシートは利用すべきであると判断した。

そこで従来の SuperSQL にスタイルシートを適用可能に拡張した。しかしこの際には多くの問題が存在した。最も大きな問題は SuperSQL はテーブルを次々と入れ子にしていくことで一つのテーブルを作っているということであった。このアルゴリズムではまず属性（値）一つに対して一つのテーブルを作成する。次にその属性を横方向に対してまとめるためのテーブルを作成し、その「セル」にそれぞれの属性のテーブルを挿入する。さらにそれを縦方向に対してまとめるテーブルを作成し、　その「セル」にそれぞれの横方向にまとめられたテーブルを挿入する。このように SuperSQL は入れ子構造になっており、出力された HTML ソースはテーブルタグが多く使われる。スタイルシートを適用する際に問題になるのがこのテーブルタグの多さである。なぜなら属性の装飾子は属性のテーブルが作成される際に読み込まれるのだが横方向や縦方向にまとめるテーブルのセルを作るタグ（<TR> や <TD>) はグルーピングが始まるとすぐに生成されてしまう。すなわち横方向や縦方向にまとめるセルのクラスを設定するためには、属性の装飾指定を「先読み」する必要があったのである。この問題の解決法として属性のテーブルの大きさを横方向や縦方向にまとめるテーブルのセルの大きさとまったく同じにして、上書きする方法を考えた。つまり正確に言えばその「セル」にはスタイルシートは適用されていない。しかしスタイルシートを適用した属性のテーブルを上からぴったりとそのセルに重ね合わせることで見た目上、あたかもセルにスタイルシートが適応されているように見せている。スタイルシートを用いることでブラウザに依存する可能性もあるが、IE6.0 では問題ないことを確認している。

スタイルシートを適用するためにまずページ全体を囲うような括弧の装飾子にスタイルファイルアドレスを引数とするような cssfile というオプションを導入し、次にスタイルシートを適用するために属性の装飾子にクラス名を引数とするような class というオプションを導入した。

4. embed 関数

4.1 背　　景

従来の SuperSQL では 1 つのクエリで複数のページを一括生成できることが大きな特徴であるが、逆に一つのページに表示したい情報は一つのクエリとして表現しなければならなかった。そのため 1 ページに表示したい内容が多岐にわたる場合や 1 ページのレイアウトが複雑な場合、クエリは長く複雑なものになることが多々あり、親和性、認知性などに問題があった。その結果、括弧の対応付けを間違えたりするなど細かなエラーなども分かりづらくなってしまっていた。これらの問題点を改善するための方法としてクエリを分割・合成を考える。

4.2 アプローチ

クエリを分割するという考え方で近いものは C などの関数である。複数回利用するものや共通箇所を関数化することでコードの理解度を高めると同時に冗長性を下げ、　その利便性を高める。また Java などではクラス化することで完全なカプセル化を実現している。これによってクラスは import 文さえ書けば自由に利用することができ、プログラムの再利用を可能にしている。

さらにクエリを合成するという考え方ではビューが近いと考えられる。ビューを利用するクエリでさらにビュー定義を書き、その結果を上位クエリで利用するという考え方である。

今回はサブクエリのように GENERATE 句の中に更なる GENERATE 句を書けるようにするというアプローチもあった。しかしこのアプローチではクエリを合成することが可能になることで 1 ページあたりの情報量やレイアウト力は向上するが、そのクエリを利用するクエリ内に記述しなければならないためクエリの長さ、冗長性などは改善することはできず、分割はできない。

分割・合成両方を可能にするため今回はクエリを外部ファイル化し、関数によってその外部クエリを読み出しその実行結果を上位部に埋め込む方法を選択し、embed 関数（埋め込み関数）という関数を導入した。これはクエリ（プログラム）を分割する（関数化する）という点で C の関数や Java のクラスの概念に近い。

しかしそれらとは違う点がある。それはカプセル化できていない点である。後述するが embed 関数では条件部を引き渡すことができるが、そのためには埋め込まれるクエリ内部の仕様を理解している必要があるのである。そのため他人の作ったクエリを即座に利用することは不可能である。しかしながら今回の提案によってクエリが短く読みやすくなることから、そのクエリがどういった動作をするのかなどを理解するためにかかる時間は大きなプログラムとは違い短いと考えられる。さらにクエリを書くためにはスキーマ設計を知らなければ結局不可能であり、この仕様によって新たに知らなければならないのはエイリアス指定など簡単に判断できるものである。そのためこのような仕様でも大きな問題はないと考えられる。

4.3 実 現 方 法

embed 関数の実現方法として embed 関数が現れたら各種引数（ファイル名、条件、ボーダーの有無）を利用して再度 SuperSQL 処理系を起動し、その出力結果（ソースコード）を上位の SuperSQL 処理系に返すという方法を試みた。

4.4 技術的問題点と解決法

始めの問題点は従来の SuperSQL の codegenerator(コード生成部）はコードを生成した後、その場で HTML ファイルを生成し、そのファイルにコードを書き込んでいたため戻り値がなかった点である。そこで本手法では新たに生成したコードをファイルに書き込まず、戻り値として扱う関数を codegenerator 部に作成した。そして embed 関数が呼び出されると、SuperSQL 処理系を再度起動するのだが、そのとき codegenerator は以前の「コードをファイルに書き込む関数」ではなく、今回新たに作成した「生成したコードを戻り値として返す関数」を利用するように変更し、その関数の戻り値として得られたコードを上位 SuperSQL 処理系のコード部に書き込むようにした。これによって embed 関数を実現した。

しかし更なる問題点が存在した。それは embed された側の装飾子が反映されないという点であった。SuperSQL の codegenerator は前述したとおり、生成したソースを生成後にファイルに書き込むのだがこのとき書き込むものはヘッダー部、コード部、フッター部と 3 種類に分離されていた。しかし今回作成した「コード部」を戻り値とする関数では装飾部、すなわちヘッダー部が上位 SuperSQL 処理系に引き渡されていないため、embed

された側の装飾子が反映されなかったのである。そこで従来の SuperSQL 処理系では行っていなかった関数、すなわちスタイル部のみを作成し、それを戻り値とする関数を新たに定義し、上位 SuperSQL 処理系の「ヘッダー部」にその戻り値を書き込むことによって、embed された側の装飾子も反映することを可能とした。

4.5 文法定義、使用方法

embed 関数の引数は最低で 1 つ、最大で 4 つである。それらは

- ファイル名
- ボーダーの有無
- where 句の左辺と条件（ < / <= / = / >= / > / <> /*LIKE*)
- where 句の右辺 (att/attString)

である。省略不可能なのはファイル名である。ファイル名は SuperSQL クエリもしくは HTML ファイルへの絶対パスである。相対パスではあってはならない。ファイルはローカル (Windows では c:など) でもオンライン (http://…) でも問題はない。そのほかの 3 つの引数に関しては省略が可能である。一つ目はボーダーの指定である。通常 embed する際には embed したテーブルの一番外側のボーダーはない方が美しい場合が多いと考えられる。なぜなら embed する側ですでにセルのボーダーが存在するにもかかわらず、embed される側で外側のボーダーを付けてしまってはその部分がボーダーが 2 重になってしまうためである。しかしながらこのボーダーを利用してテーブルをより美しく見せる方法も工夫次第ではあると考えられるため、引数に「border」を追加した。次の引数は embed される SuperSQL クエリに条件を引き渡すための引数 2 つである。まず一つ目は where 句（条件部）の左辺と条件 (< / <= / = / >= / > / <> /*LIKE*) を引き渡すための引数、「where」である。状況によっては embed する側からされる側へある条件を引き渡したい場合も考えられる。例えばある社員の ID をされる側に引き渡すことでその社員のデータのみを embed する、といった具合である。そうしたことを可能にするために引数「where」を追加した。「where」には SQL の where 句（条件部）の左辺と条件までをダブルクォート (") で括って指定する。さらに条件を引き渡すためには条件の右辺も引き渡す必要がある。その引数が「att/attString」である。「att」と「attString」の違いは引き渡す値が int 型か String,Timestamp,char 型かの違いである。引数には属性名を記述する。

4.6 クエリの再利用

embed 関数を利用することにより、クエリを再利用することが可能となる。例えば非常に簡単なクエリ、車のリストを全て表示するようなクエリを作成しておく。そして上位クエリでそのクエリを利用するのだが、その際条件部を引き渡すことである特定のメーカーの車のリストのみを表示するクエリとして使用できる。そのため単純なクエリが embed する側によって多種多様に変化するのである。また同一の RDB の表から多くの違ったデータを引き出す場合、多くのエイリアス指定を利用することになる。これは非常に見づらく初心者には分かりづらいものである。しかし embed 関数を利用することで同じ表から引き出す場合には embed するようにすればこういった問題も解決することができる。特にフレームが推奨されなくなり、メニューバーなどをテーブルタグなどを利用して作成する必要が増えたことで、HTML ファイルに共通部分が増えることになった。それらを各クエリに書くのは冗長であり、クエリを再利用できるようになることは有意義であると考えられる。

4.7 使 用 例

図 4.7 を作成するために必要なクエリの一部を以下に表示する。最上位部のクエリは embed.sql で様々なクエリが embed されている。list1.sql や menu1.sql はその一部である。list1.sql は非常に簡単なクエリであるが、embed 関数の where 句の引渡しによってそれぞれ用途に応じたクエリとなる。

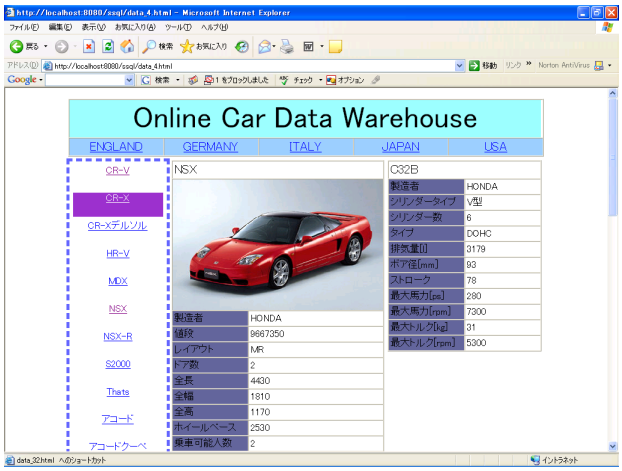


図 1　embed 使用例

list1.sql

```
GENERATE HTML
{    {
   [sinvoke((asc1)ca.name@{class=list},
file="http://localhost:8080/ssql/data.sql",
att=ca.id)]!
}@{tableborder=0}
}@{cssfile=demo.css}
FROM car ca;
```

menu1.sql

```
GENERATE HTML
{      {
   "Online Car Data Warehouse"@{class=title}!
   [sinvoke(co.country@{class=menu},
file="http://localhost:8080/ssql/menu.sql",
att=co.country)],
}      }@{cssfile=demo.css}
FROM company co
```

embed.sql

```
FOREACH (ca.id)
GENERATE HTML
{  {
  { embed(file="c:\SSQL\demo\menu1.sql")} !
  {
    { embed(file="c:\SSQL\demo\list1.sql",
where="ca.company=",
att=co.id,border=1)@{class=colist} },
    { embed(file="c:\SSQL\demo\car.sql",
where="ca.id=",att=ca.id)@{width=200} },
    { embed(file="c:\SSQL\demo\engine.sql",
where="e.id=",att=ca.engine)@{width=200}}
  }
}@{tablealign=center,tableborder=0}
}@{cssfile=demo.css}
FROM car ca, company c
WHERE ca.company=co.id
```

5. FORM

5.1 背　　景

データベースの内容を得るためには様々な方法がある。コマンドプロンプトからデータベースを立ち上げて閲覧する方法が最も単純ではあるが、その方法では SQL などの専門的な知識が必要で一般的な利便性は低い。またその出力は無機質で親和性が高いとは言い難い。そのためそのような専門的な作業は専門の SE などが行い、一般のユーザがデータベースの内容を閲覧するためには SE がアプリケーションを開発し、そのアプリケーションを利用して内容を閲覧するのが一般的である。アプリケーションは通常 Java,php,Perl などのプログラミング言語を用いて作成される。しかしながらそういったアプリケーションの作成には必要な専門知識が幅広く、作成できる人間は限られている。そこで考案されたのが SuperSQL である。しかしこのままではページの生成はできるもののフォームから与えられた条件に対応する「検索」などを行うためには不十分である。通常「検索を行うプログラム」はクエリ生成部、クエリ実行部、HTML ソース生成部から成る。そのうちの後半 2 つ、クエリ実行部とソース生成部に関しては SuperSQL によって既に実現されている。そこでフォームから SuperSQL 処理系へ条件部（where 句）を引き渡すことでクエリを完成させ、SuperSQL を利用し動的に出力を得ることを考え実装を行った。これにより Web アプリケーションの大部分を占めるデータ閲覧型アプリケーションを SuperSQL クエリと HTML によるフォーム部によって簡易的に作成することが可能になる。

5.2 サーブレットの実装

各種必要な値をフォームから受け取り条件部を生成し、それに基づいて SuperSQL 処理系を起動する。必要な値とは

- ホスト名
- ユーザ名
- データベース名
- SuperSQL クエリファイルアドレス
- 各種条件

である。受け取り側（サーブレット側）はリクエストパラメータメソッドによって各種必要な値、アドレスを取得する。ページ作成に不可欠なホスト、ユーザ名、データベース名を記述した設定ファイルアドレスと SuperSQL クエリファイルアドレスが受け取れなかった場合、エラーメッセージを書き出し、システムは終了する。SuperSQL 処理系を起動する際、設定ファイル、条件を引数として起動する。条件を引数として起動する方法は既に実装済みであったため、それを利用した。この際、従来の SuperSQL とは違いサーブレットでは HTML ファイルを作成することはしないこととした。このため SuperSQL の codegenerator(コード生成部) に手を加える必要がある。なぜなら従来の SuperSQL の codegenerator はコードを生成後、HTML ファイルに書き込んでしまうからである。ここでは前述した embed 関数実装時に「コード部」を戻り値とした関数を既に実装していたため、これを利用した。しかし 4. 章でも述べたが SuperSQL 処理系では HTML コードをヘッダー部、コード部、フッター部と分けて生成していた。そこでヘッダー情報を戻り値とする関数を新たに codegenerator に作成した。そしてそれら 2 つの戻り値を PrinterWriter によって出力することで、結果を表示する。PriterWriter によって出力することによって、この出力はその場限りとしており、キャッシュに残ることやましてファイルとして生成され保存されることはない。なぜなら同じ条件で検索することは稀であると考えられるためである。また従来の SuperSQL では出力結果が 0 の場合、エラーメッセージを表示し、SuperSQL 処理系を終了していたが、新しいシステムでは「NO DATA FOUND」と出力するコードを出力するように変更した。さらに問題点は存在した。それは今までは設定ファイルや SuperSQL クエリファイルというのは「ファイル名」で SuperSQL 処理系に引き渡していた。しかし今回の場合オンラインのため「URL」になってしまったのである。そのため SuperSQL 処理系内部では FileReader を利用していたが、これでは問題が生じてしまった。そこで SuperSQL クエリの読み出し方法と設定ファイルの読み出し方法に手を加えた。まず SuperSQL 処理系を起動するときに、オンラインで起動した場合にはそれ相応のフラグを立てる。そのフラグがたっていなければ従来どおり FileReader を利用して読み込む。そのフラグがたっていれば新たに作成した関数を利用し、SuperSQL クエリと設定ファイルを読み込む。新たに作成した読み込み用関数では FileReader の代わりに DataInputStream を利用している。正確には URL を取得し、URLConnection によって指定された URL との接続を確立し、DataInputStream を利用してデータを読み込む。これによって、SuperSQL クエリと設定ファイルをオンラインに置いておくことができるようになった。

5.3 フォーム作成方法

フォーム側では前述した必要な値を受け渡す必要がある。そのため多くの $< input >$ タグのタイプが「hidden」となってしまうが、これらの情報は一つとして欠かすことができない。また実際に条件分岐などをプログラムする場合と比べれば十分簡単であると考えた。まずフォームである以上 action 先を指定しなければならない。action 先のアドレスはこのサーブレットのアドレスとなる。つぎにホスト名、ユーザ名、データベース名を記述した設定ファイルを作成する。ファイル名は任意だがファイルの拡張子は「.ssql」でなければならない。

config.ssql

```
host=localhost
user=tk
db=tk
```

設定ファイルを受け渡す際の $<$ *input* $>$ タグの name 属性は「configfile」で、value 属性にそのアドレスが入る。さらに SuperSQL クエリファイルのアドレスも受け渡す必要がある。この際の name 属性は「sqlfile」で、value 属性にそのアドレスが入る。これらの他に受け渡す必要があるのが「条件」である。入力方法はテキスト、チェックボックス、ラジオボタン、セレクトボックスが代表的である。条件部には大きく分けて 3 つの部分に分けることができる。右辺、左辺、そして条件である。それぞれ受け渡すことになる。さらにここではさらに右辺の属性の型 (int 型、char 型、etc) も指定する必要があるため、必然的に 1 つの条件入力にたいして 4 つ以上の $<$ *input* $>$ タグが必要になる。ただしチェックボックスの場合に限り例外的に $<$ *input* $>$ タグは 1 つである。まずテキストの場合について述べていく。まず左辺の受け渡しには「hidden」タイプを使用し、name 属性を cond_name(i) とする。このとき (i) には 1 以上の数字が入る。条件が増えるにつれてその番号は 2，3，4 ･･･ と増加していく。そしてその value 属性に左辺の属性名を記入する。次に条件の受け渡しには同じく「hidden」タイプを使用し、name 属性を cond(i) とする。そしてその value 属性に (<= / < / = / > − >= / <> /*LIKE*) のいずれかを記入する。次に右辺の属性の型を受け渡す。これまでと同様に「hidden」タイプを使用し、name 属性を value_type(i) とする。そしてその value 属性に「int」、「String」、「ORint」、「ORString」を指定する。「String」には数字以外の型、すなわち timestamp 型や char 型も含まれる。OR の場合はこの条件を AND ではなく OR で連結する。最後に右辺、すなわち値の受け渡しを行う。この場合は入力をしてもらうため当然「hidden」タイプではなく text タイプを用いる。そしてその $<$ *input* $>$ タグの name 属性を value(i) とする。ラジオボタン、セレクトボックスも多少の工夫で作成することができる。ただしチェックボックスに関しては例外で、cond_name(i), cond(i), value_type(i)　は省略し、value(i) に全ての条件部、すなわち右辺、条件、左辺を記述する。そして最後には cond_name(i) の value 属性として「exit」を受け渡すことで入力部の終了を受け渡す。(i) の値を 1 から増やしていくことで条件の数は無限に増やすことが可能である。

5.4 使 用 例

クエリ例

```
GENERATE HTML
{  {
[m.name,m.salary]!
}@{tablealign=center}  }
FROM member m
```

フォーム例

```
<<HTML>
<head>
</head>
<body>
<form method="post"
action="http://localhost:8080/ssql/
servlet/supersql.form.FormServlet">

<input type="hidden" name="configfile"
value="http://www.db.ics.keio.ac.jp
/~tk/config.ssql">
<input type="hidden" name="sqlfile"
value="http://www.db.ics.keio.ac.jp
/~tk/form.sql">

<input type="hidden" name="cond_name1"
value="m.name">
<input type="hidden" name="cond1" value="=">
<input type="hidden" name="value_type1"
value="String">
氏名<input type="text" name="value1"><BR>

<input type="hidden" name="cond_name2"
value="m.salary">
給料<input type="text" name="value2">
<input type="radio" name="cond2" value=">=">
以上
<input type="radio" name="cond2" value="<=">
以下<br>
<input type="hidden" name="value_type2"
value="Int">

<input type="checkbox" name="value3"
value="m.grade = 'B4'">B4

<input type="hidden" name="cond_name4"
value="exit">
<input type="submit" name="submit" value="送
信">
</form>
</body>
</HTML>
```

フォームのソースについて簡単に解説する。まずフォームのアクション先を指定し、設定ファイル・クエリファイルのアドレスを引き渡す。1 つ目の条件は「name=' 入力値'」となっている。また 2 つ目の条件は「salary >= 入力値」もしくは「salary <= 入力値」となるようにラジオボタンとテキストボックスを

用いて作成されている。さらに 3 つ目の条件はチェックボックスを用いて作成されている。最後に 4 つ目の cond_name として exit が引き渡され条件部の終了を明示している。実際に実行されるクエリは引き渡されたクエリ＋引き渡された条件部（WHERE 以下）となる。この検索結果を表示する方法として、SuperSQL クエリを利用する。

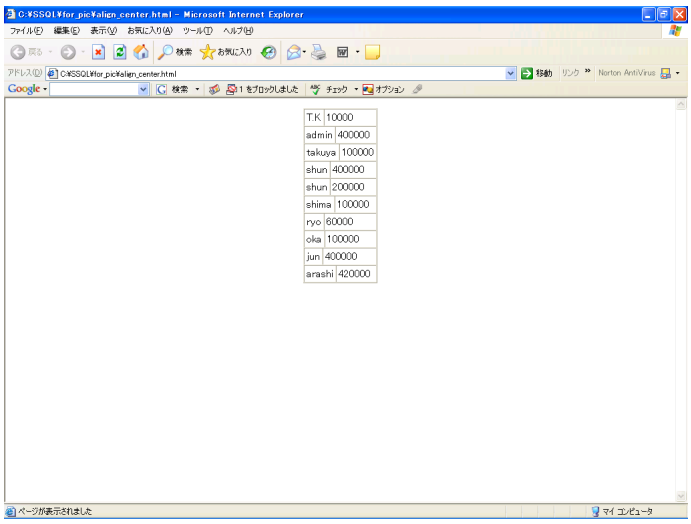


図 2　フォーム出力例

6.　検討・アンケート結果

本研究の結果・検討のためにアンケートを実施した。被験者数は本研究の本質を知るために十分な数とし、SuperSQL の詳しい人から知らない人まで多岐にわたる。ここではページの関係上有用性を確かめるために十分な回答のみを掲載する。まず CSS を適用していないページと CSS とフレームを利用したページを見比べてもらった。その上でまずどちらのほうが利用しやすいかを答えてもらった。その結果が図 3 である。約半数がフレームと CSS を利用したページのほうが利用しやすいと回答している。それに対し 2、1 は 0 %となっており、フレームと CSS を利用することは非常に利便性を高める上で効果的ということが確認できる。さらにページ内容をどちらのほうが理解しやすいかを答えてもらった。その結果が図 4 である。その結果約 6 割がポジティブな回答をしている。ネガティブな回答をしたのはわずかに 8 %に止まっている。ネガティブな回答のコメントを見ると

- タブブラウザなので新たなウィンドウが開くのが嫌
- 新たなウィンドウが開くのはページ遷移が変

という意見が多数でに CSS を否定したものは存在していなかった。

次に従来手法と提案手法を用いて同じページを作成した場合のクエリの総行数を比較する。縦軸に行数、横軸は共通箇所の数に比例する値である。例えば、1 の場合には 3 種類のページ中 2 つのページに 1 箇所のみ共通箇所がある、2 の場合には全てのページに共通箇所が 1 箇所がある、6 の場合には全てのページに共通部分 3 箇所ある、という風である。図 6. を見ると、提案手法の方が従来手法よりも少ない行数となっており、また共通箇所が増えれば増えた方が総行数の差が大きくなっていることが分かる。これは従来手法では共通箇所があっても各クエリにすべて書かなければならなかった。それに対して提案

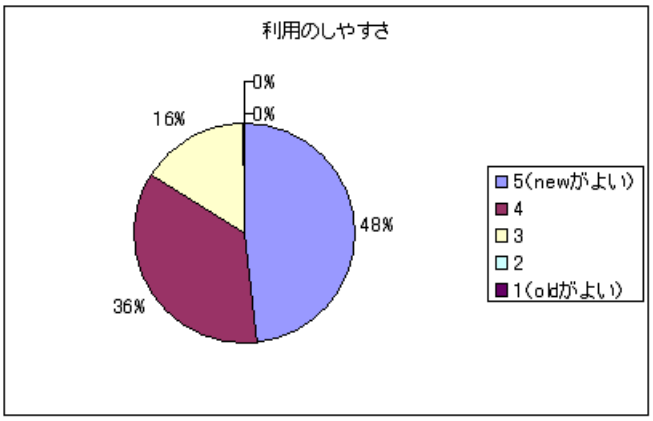


図 3　どちらが利用し易いか

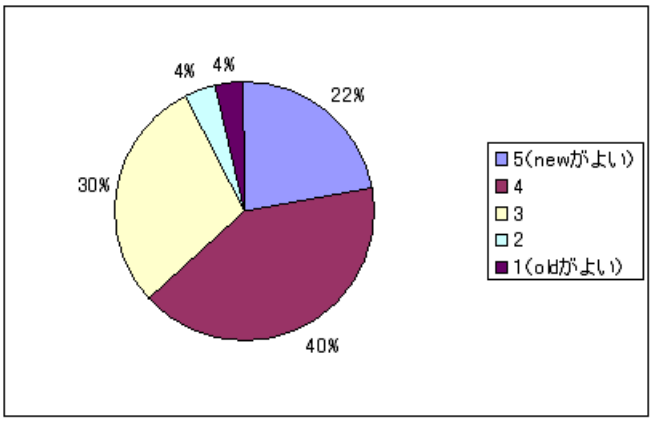


図 4　どちらが内容理解しやすいか

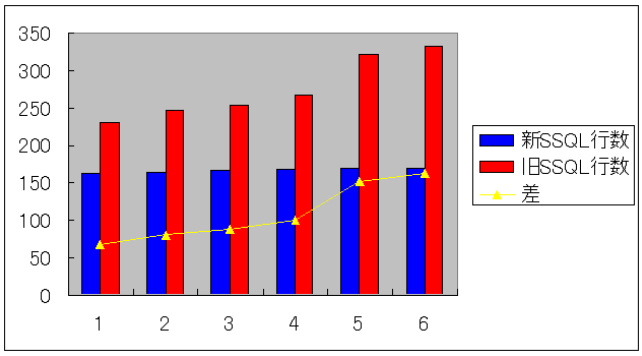


図 5　従来手法と提案手法の総行数の比較

手法では共通箇所は一つのファイルとして書いておき、それを利用する際には呼び出す embed 関数を 1 行記述するだけで済むためである。すなわちクエリを再利用できているということである。

さらにクエリの細分化についてアンケートを実施した。まず簡単なクエリを組み合わせ、複雑で大きなクエリを作成できるようになることでページのレイアウトなどの自由度が上がるか … という質問をし、その結果が図 6 である。結果約 4 割が自由度が上がると答えている。中間的な意見を含めれば 8 割になる。このアンケートを実施した際にはあまり効果的な例を挙げられていなかったため若干結果が悪いものの十分その自由度が上がるという主張に関して理解を得られていると考えられ

る。またクエリの再利用は便利か … との問いに対する結果が図 7 である。約 7 割がポジティブな回答をしている。再利用の有効性を十分に証明できていると考えられる。

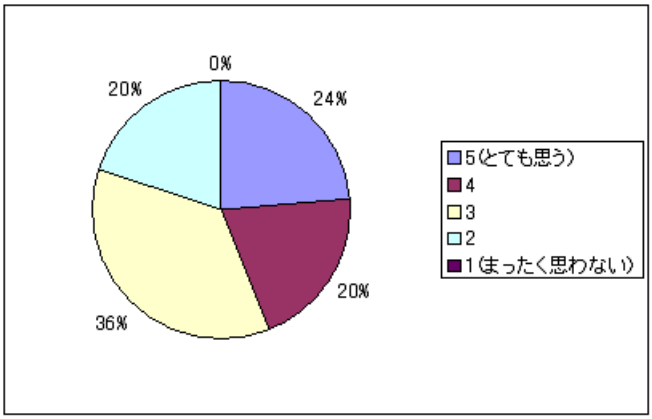


図 6　embed によって自由度が上がるか

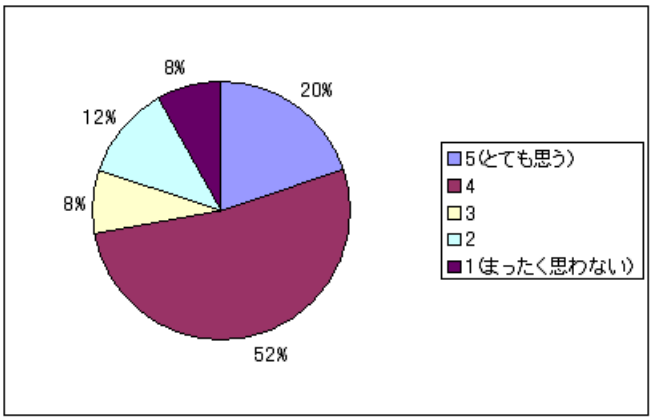


図 7　クエリ再利用は便利か

さらに飛行機検索システムを PHP で作成した場合と提案手法で作成した場合の総行数を比較した。その結果が図 6. である。その結果、提案手法の方が PHP と比べて 60 %程度削減できていることがわかる。これは作業量の低下を示している。

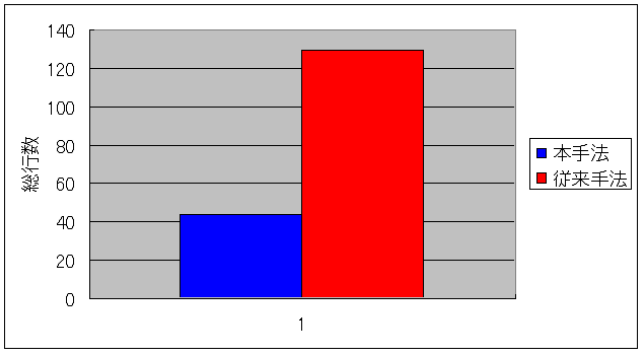


図 8　php と提案手法の総行数の比較

加えて飛行機検索システムをを本手法によって作成してもらった。その結果、php などの言語と比べて簡単になったかどうかを回答してもらった。その結果が図 6. である。その結果約 4 割が簡単になったと答えている。さらに中間的な意見を合わせると実に 9 割が簡単になったと答えている。この結果は本手法の有用性を十分に示していると考えられる。

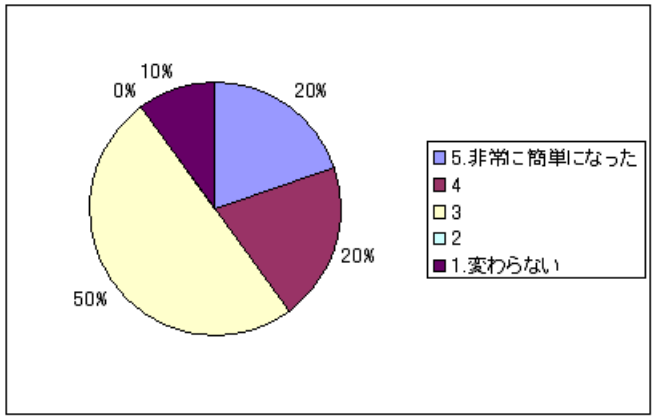


図 9　検索システム作成は簡単になったか

7. ま と め

本研究では SuperSQL クエリを細分化し、小さく簡単なクエリを組み合わせることで一つの大きなクエリを成形し、ページを出力する方法を提案した。また同時にクエリを再利用し、より少ないクエリで多くの出力、情報を得る方法も提案した。さらにフォームを用いることで SuperSQL 処理系を遠隔に起動する方法を提案した。

文　献

[1] SuperSQL: http://ssql.db.ics.keio.ac.jp/

[2] M. Toyama, “SuperSQL: An Extended SQL for Database Publishing and Presentation”, *Proceedings of ACM SIGMOD ’98 International Conference on Management of Data*, pp. 584-586, 1998

[3] 遠山 元道 :『ターゲットリストの拡張によるデータベース出版と概視の実現』、信学技報、Vol.93,No.152、P79-88、電子情報通信学会、1993

[4] 岡部　康矢: 『SuperSQL における HTML 出力の高品質化』、学士論文、慶應義塾大学理工学部情報工学科、2002

[5] 寺田 純子: 『SuperSQL クエリにおける条件付装飾指定の実現と外部ファイル化』、学士論文、慶應義塾大学理工学部情報工学科、2004