SuperSQL処理系におけるプラグイン関数および局所的整列の実現

中道　亮† 　金　翰基†† 　遠山　元道†

† 慶應義塾大学 理工学部 情報工学科　〒 223–8522 横浜市港北区日吉 3–14–1
†† 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 横浜市港北区日吉 3–14–1
† 慶應義塾大学 理工学部 情報工学科　〒 223–8522 横浜市港北区日吉 3–14–1
E-mail: †{ryo,hanki}@db.ics.keio.ac.jp, ††toyama@ics.keio.ac.jp

あらまし　SuperSQL の特徴はワンソース・マルチユースや出力結果の構造化である。したがって従来の SQL とは異なり、ユーザの多様な要求仕様に対応するための機能の拡張性が重要となってくる。しかし、拡張に伴う言語仕様の変更は極力回避し、処理系の内部仕様を変更することなく、拡張モジュールを用意するだけで対応できることが望ましい。そこで本研究では、関数の追加による機能拡張を目標とし、ケース分析により、集約型関数とスカラー関数の 2 タイプについてプラグイン機構の設計をし、従来の SuperSQL 処理系の基本関数を含め、すべてのプラグイン化が実現し、将来に渡る拡張性を実現した。また、プラグイン開発の自由度を向上させるため、あるレベルでの演算結果をローカル変数として保持する assign 関数を実装し、それより下の次元で自由に参照できることを可能とし、SuperSQL 処理系に代入という新しい概念を導入した。

キーワード　SuperSQL、DB 言語、問合せ処理

Implementation of plug-in function and local alignment in SuperSQL

Ryo NAKAMICHI†, Hanki KIM††, and Motomichi TOYAMA†

† Department of Information and Computer Science, Faculty of Science and Technology,
Keio University
Hiyoshi3–14–1, Kouhoku-ku, Yokohama-shi, 223–8522 Japan
†† School for Open and Envionmental Systems, Graduate School of Science and Technology,
Keio University
Hiyoshi3–14–1, Kouhoku-ku, Yokohama-shi, 223–8522 Japan
† Department of Information and Computer Science, Faculty of Science and Technology,
Keio University
Hiyoshi3–14–1, Kouhoku-ku, Yokohama-shi, 223–8522 Japan
E-mail: †{ryo,hanki}@db.ics.keio.ac.jp, ††toyama@ics.keio.ac.jp

Abstract　SuperSQL can generate various kinds of structured publication and presentation documents. Therefore, it differs from established SQL, Extendibility of function for corresponding to various demands of user is important. But there should try not to change of the language specification accompanying extension, and it is desirable that it is realizable only by preparing modules for functionality expansion, without modification of the original source code. So, in this paper, we suggest the plug-in mechanism for aggregate function and scalar function. The result is that we could change established standard functions of SuperSQL into plug-in and it is realized extendibility over the future in SuperSQL. In addition, we implement assign function that holds an operation result in a dimension as a local variable for improvement in the ability to define of plug-in. The result is that it is realized that use the local variable at a lower dimension, and the new concept of substitution was introduced into SuperSQL.

Key words　SuperSQL, DB Language, Query Processing

1. はじめに

SuperSQL [3] [4] の特徴はワンソース・マルチユースと出力結果の構造化であり、その特徴を活かす為の機能が大きく 2 つに分類出来る。1 つは前者の特徴から由来する各メディアに特化した機能であり、もう 1 つは後者の特徴から由来するグルーピングされたタプルに対する集約系演算機能であり、すでに幾つか実現されている。しかし、まだまだ各ユーザによって異なる細かい要求仕様を満たすことが出来るとは言い難い。また、本質的では無く過渡的ではあるが、現在において SuperSQL クエリ中では演算子を記述することが出来ない為、属性値を用いた単純な四則計算を行うことも出来ない。

しかし、現在の SuperSQL 処理系においては、機能を追加するには処理系内部に手を加えるしかない状態である。しかし、ソースコードを持たない一般ユーザにとって機能を追加することは不可能であり、開発者に依頼するしかない。しかし、各ユーザから寄せられる様々な依頼を開発者が一手に引き受けることは非現実的である。また、仮に各ユーザによって機能を追加出来たとしても、言語仕様が統一出来なくなる恐れがある。

このような問題を解決するため、本研究では関数の追加による機能拡張を目標とし、ケース分析により、集約型関数とスカラー関数の 2 タイプについてプラグイン機構を設計し、言語仕様を変えることなく必要な機能の追加を容易に実現出来ることを可能とした。プラグイン開発の自由度を向上させるため、あるネストレベルでの演算結果をローカル変数として保持する assign 関数を実装し、それより下のレベルで参照できることを可能とし、SuperSQL 処理系に代入という新しい概念を導入した。

以下、本稿の構成を示す。まず、2. 章で本研究におけるプラグイン関数機構の概要について述べる。次に、3. 章では、プラグイン関数の実際の定義例を示す。そして、4. 章ではシステムの内部仕様について述べ、5. 章では、局所的整列について述べる。そして、実際に実行した結果や、評価、検討を 6. 章で述べ後、最後に 7. 章で結論を述べる。

また、本稿では以下のようなプロ野球選手の個人データに関するデータベースの例を用いる。

選手 (名前 varchar, チーム varchar, 身長 int, 体重 int)

2. 本システムの概要

2.1 プラグイン関数の使用方法

本研究では、ユーザは要望から実現までにしなければならないことはプラグインの開発とその定義のみである。また、プラグイン関数の定義については、集約関数、スカラー関数の 2 通りについて定義することができる。例えば、「平均身長を求めたい」といった集約演算を行いたいといった要望、「身長と体重を用いて各個人ごとに BMI 値を求めたい」といったスカラー演算を行いたいというユーザの要望があったとする。このときユーザが行わなければならないことは、前者であれば、「平均値を求める集役関数」のプログラムを記述し、その関数仕様を定義する、後者であれば「BMI 値を求めるスカラー関数」のプログラムを記述し、その関数仕様を記述するのみである。例えば前者を avg、後者を BMI と定義した場合、クエリ中で両関数が使用できるようになり、avg の引数に身長、BMI の引数に身長と体重を与えることによって要望に応じた出力結果を得ることができる。BMI 値などはクエリ中に演算子が記述できればこの様に関数を定義する必要が無いが、例えばプラグイン関数にすることで、ユーザが BMI 値が基準値を越えている人に対して警告の意味で BMI 値の前に「！」を表示させたいと思ったときなどは関数のプログラム中に記述すれば容易に実現出来るようになる。この様な表示方法は、プラグイン関数以外では不可能である。例えば BMI と avg を定義した場合、以下の図 1 の様なクエリが記述可能となる。

```
1  /* sample1.ssql */
2  GENERATE HTML [
3      fp.team ,
4          [ fp.name, BMI(fp.height, fp.weight) ]! ,
5      avg(fp.height)
6  ]!
7  FROM field_player fp
```

図 1　サンプルクエリ 1

以下、図 1 のクエリについて詳しく説明する。まず 3 行目で選手をチーム毎にグルーピングし、、4 行目で各選手ごとに BMI を表示している。そして 5 行目でチームの平均身長を表示している。

2.2 定義における問題点

プラグインを定義するにあたって、ユーザから他のプラグイン関数によって処理された結果を他のプラグイン関数において参照したいという要求が出てくると考えられる。それを解決する手段の一つが入れ子型のプラグインであるが、本研究では、外側の関数のプログラム中で引数として与えられた関数を呼び出すという手法を取っている為、結局最終的にはデータベースから取得した属性値に対して処理を行うことになる。その為、SuperSQL の出力結果を構造化出来るという特性上、値の参照が出来ない場合が存在する。

2.3 値の参照パターン

まずプラグインの入れ子のパターンを考える。SuperSQL では出力結果を構造化出来るという特性を持つので、プラグイン関数の入れ子には以下の 4 パターンが考えられる。

（1） スカラー関数の引数にスカラー関数
（2） スカラー関数の引数に集約関数
（3） 集約関数の引数にスカラー関数
（4） 集約関数の引数に集約関数

ここで SuperSQL の特性ゆえに最大の問題となってくるのはそれぞれの関数におけるグルーピングの基準の違いである。1 は一致しているが、2, 3, 4 は外側の関数と引数の関数のグルーピングの基準が異なっている。1 については一つの関数にまと

めてしまえばよいため、あまり利用価値がないと考えられる。そのため、残り 3 パターンにおいて各パターンが求められる状況を考えてみる。まず、2 のパターンは、チームの平均身長と個人の平均身長との差を求めたいといった、「上位レベルで得られた演算結果を下位レベルで用いたい」という状況で求められ、3 のパターンは、個人の BMI 値を用いて BMI のチーム平均を求めたいといった、「下位レベルの演算結果を上位レベルで用いたい」という状況で求められる。また、4 のパターンにおいては、各チームの平均身長の最大値を求めたいといった「下位レベルの演算結果を上位レベルで用いたい」場合と、全チームの平均身長とチームの平均身長の差を求めたいといった「上位レベルで得られた演算結果を下位レベルで用いたい」場合と両方の場合で求められる。この中でも 3 のパターンの「下位レベルの演算結果を上位レベルで用いたい」場合、関数の入れ子を用いて外側の関数のプログラム内に引数の関数を呼び出す記述をすることによって実現可能であるが、2,4 のパターンにおける「上位レベルで得られた演算結果を下位レベルで用いたい」場合、および、4 のパターンにおいては「下位レベルの演算結果を上位レベルで用いたい」場合は実現不可能である。「上位レベルで得られた演算結果を下位レベルで用いたい」場合の理由は、下位レベルの関数の実行時に渡されたタプル数では上位レベルの関数を実行出来ない為である。例えば、チームの平均身長と個人の平均身長との差を求めたいといった場合、差を求めるスカラー関数を実行した時点で送られてくるタプルは当然 1 タプルだけであるため、チーム平均を求めることは不可能であるといった具合であり、受け取ったタプルで引数の関数の処理結果を求めることが出来ないのである。よって、「上位レベルで得られた演算結果を下位レベルで用いたい」場合を解決するアプローチを別に考える必要がある。

また、4 のパターンにおける「下位レベルの演算結果を上位レベルで用いたい」場合についてのみ本研究におけるプラグインの処理アルゴリズム上、現段階では不可能である。この理由については 6. 章の検討で詳しく述べる。

2.3.1 演算結果の参照方法

プラグイン処理の特性上、必ず上位レベルから処理されて行くため、下位レベルのプラグインが実行される時点では上位レベルの演算結果はすでに得られている。よって、下位レベルのプラグインの引数においてその演算結果を参照出来るようにすればよく、つまり、下位のプラグインの引数で演算結果を参照したい上位のプラグインを指定してその演算結果を参照出来るようにすれば良いことになる。そこで、一旦ローカル変数として演算結果を保持しておいて下位レベルでそのローカル変数を用いて参照するという方法が挙げられる。しかし、SuperSQL の特性上、値そのものをローカル変数を保持しておいてそれを参照するという方法は不可能である。なぜなら、参照したいプラグインがあるレベルでのグルーピングは複数の場合が多く、例えば、チームであればグループは 12 グループあり、平均身長は 12 通りある為、同一のローカル変数名で保持出来ない為である。よって、本研究では自らのタプル内の値を参照するという方法を採用した。上位レベルのプラグインが処理された段階でそこに該当する属性値はその演算結果に書き換えられている為、それを参照すれば複雑な処理をしなくてもよいことになる。指定方法については、プラグイン関数自体にローカル変数名を与え、その演算結果があるタプル内でのポジションのペアリストを作成し保持しておき、下位レベルで参照する際は、引数に参照したいローカル変数を用いる。そしてプラグインの処理の前段階でそのリストに問合せ、引数として与えられたそのローカル変数が参照すべきポジションの値を代入すれば最も簡単な処理かつ、ユーザがクエリを記述する際にも、参照したいプラグインのレベル等を気にしなくてもよく、直観的にわかりやすいと思われる。そこで、本研究ではこの方法を実現し、プラグイン定義の自由度を向上させるために assign 関数と呼ばれる関数を導入した。

```
1  /* sample2.ssql */
2  GENERATE HTML [
3      fp.team ,
4      [
5        fp.name, fp.height, diff(fp.height, teamavg)
6      ]! ,
7      assign(avg(fp.height), teamavg)
8  ]!
9  FROM field_player fp
```

図 2　サンプルクエリ 2

2.4 assgin 関数の定義

assign 関数の定義は以下のようになる。

$assign(<$ プラグイン関数 $>, <$ ローカル変数名 $>)$

第 1 引数には下位レベルで参照したいプラグイン関数、第 2 引数にはその値を保持しておくためのローカル変数名を記述する。例えば、

$assign(avg(fp.height),\ avg)$

と記述すれば、assign 関数が記述されているレベルで avg 関数が実行され、その結果が avg として保管される。

2.5 assign 関数の使用方法

assign 関数を用いて演算結果を参照する場合、クエリは以下の図 2 の様に記述すればよい。ここで使用している $diff$ は、第 1 引数と第 2 引数の差を返す関数である。avg の機能に関しては、前述したものと同じである。

以下、図 2 について詳しく説明する。まず 3 行目で選手をチーム毎にグルーピングしており、そして 7 行目で assign 関数を用いてチームの平均身長を求め、それを $teamavg$ というローカル変数として保持している。そして、5 行目において、$diff$ を用いて各選手の身長とその選手が所属するチームの平均身長との差を表示している。

3. プラグイン関数の定義方法

ここではまず、本研究で用意したプラグイン関数定義用ファ

イルを示し、その定義用ファイルを用いて実際にプラグイン関数を定義した例として各選手の身長と体重を用いて各選手ごとに BMI 値を求める BMI 関数を示す。

3.1 プラグイン定義用のテンプレート

プラグイン定義用のテンプレートファイルを図 3 に示す。ユーザはこのテンプレートファイルを用いてプラグインを定義する。

```
1  /* template.java */
2  package supersql.plugin.plugins;
3
4  import supersql.extendclass.ExtList;
5  import supersql.plugin.PluginSet;
6  import supersql.plugin.GetArg;
7
8  public class template extends PluginSet{
9   public String exe(ExtList buffer, ExtList target){
10 /*
11  *
12  *  プラグイン処理記述部分
13  *
14  */
15    String result = '演算結果';
16    return result;
17  }
18 }
```

図 3　template.java のソースコード

以下、図 3 について詳しく説明する。まず、4〜6 行目で必要最低限のインポートを行っており、後は必要に応じてクラスを追加していけばよい。続いて 9 行目を見ると分かるように、プラグインの戻り値は String 型でなければならず、15,16 行目のように、演算結果を一旦 String 型にして返す記述が必要である。これは、内部処理系に手を加えずにプラグインのインスタンス自動生成を行えるようにする為である。また、引数の buffer はタプル (タプル集合) であり、target は引数のタプル中におけるポジションリストである。引数をタプル (タプル集合) として渡す理由は、SuperSQL 処理系内部では、データベースに格納されている型がどのような型であってもデータベースから取得した時点でタプルは ExtList に格納され、元々の型が情報が失われるため、「この属性値の型は〜だ」という判定が出来ないためである。そして、戻り値を String 型にした理由としては、タプルを扱うために使用されている ExtList は、例えば数値型を数値型としてそのまま格納することが出来ず、結局一旦 String 型などに変換して格納しなければならないため、String 型が一番扱いやすい為である。

3.2 BMI 関数の定義

図 4 は BMI のソースコードである。

以下、図 4 について詳しく説明する。実際の処理は 17〜18 行目のみであり、それ以外はプラグイン定義の際の規則を満たす記述である。また、ここで着目すべきとこは 13〜16 行目である。これはプラグインの引数の値をタプル中から参照して取り出す記述だが、引数の値を取り出す為に GetArg クラスを用いなければならないという記述規則がある。これは引数の値をタプル中から取得する為に準備されたクラスであり、また戻り値は String 型である。GetArg クラスの getArg メソッドを呼び出し、タプルである buffer, target に加え、今取り出そうとしている値のプラグイン内での引数としての順番、つまり第 1 引数であれば 0, 第 2 引数であれば 1 というを数字を引数として渡す。BMI は引数を 2 つとる関数で、第 1 引数としてに身長、第 2 引数として体重を受け取る。よって、11〜13 行目の身長を取り出す為に呼び出した getArg メソッドの第 3 引数には 0, 14〜16 行目の体重を取り出す為に呼び出した getArg メソッドの第 3 引数には 1 を与えている。また、戻り値は String 型の為、Double 型に変換している。

```
1  /* BMI.java */
2  package supersql.plugin.plugins;
3
4  import java.text.NumberFormat;
5  import supersql.extendclass.ExtList;
6  import supersql.plugin.PluginSet;
7  import supersql.plugin.GetArg;
8
9  public class BMI extends PluginSet{
10  public String exe(ExtList buffer, ExtList target){
11    double height =
12      Double.parseDouble
13       (GetArg.getArg(buffer, target, 0));
14    double weight =
15      Double.parseDouble
16       (GetArg.getArg(buffer, target, 1));
17    double BMI =
18      weight / ((height / 100) * (height / 100));
19    NumberFormat formatter =
20      NumberFormat.getNumberInstance();
21    formatter.setMaximumFractionDigits(4);
22    formatter.setMinimumFractionDigits(2);
23    String result = formatter.format(BMI);
24    return result;
25    }
26 }
```

図 4　BMI.java のソースコード

4. システムの実装

本研究では、現在の SuperSQL の内部処理系に新たにプラグイン処理を行うためのクラスおよびパッケージを追加することによってプラグインの使用を可能とした。以下、まず SuperSQL の内部処理系を示し、本研究においてどのようなクラスを追加したかを述べる。次に、プラグイン関数のみと assign 関数を含む場合との 2 通りについて実際のクエリを用いてシステムの処理アルゴリズムを説明する。

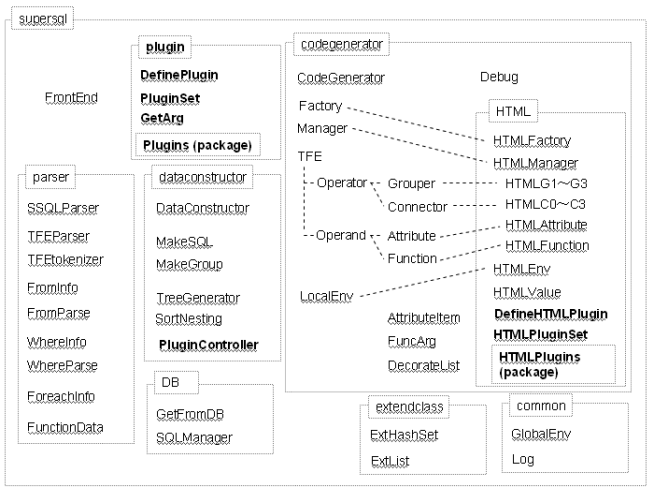

図 5　クラス、パッケージ追加後の構造

4.1　現在の SuperSQL の内部処理系

現在 SuperSQL 処理系の構造は java を用いてそれぞれが独立した役割を持つ 9 つのパッケージからなる処理系として実装がなされている。本研究では、まず演算系プラグイン関数を定義し使用するために以下のクラスとパッケージを追加した。追加した場所等は詳しくは図 5 に示す。また、このクラスやパッケージはあくまで演算系プラグイン関数を処理するためのものであり、各メディアに特化した関数を処理する為には、図 5 を見ると分かるように、PluginController クラス以外の各メディア専用クラスとパッケージを追加する必要がある。

- PluginController.java：プラグインの処理用クラス。
- DefinePlugin.java：プラグインの定義用クラス。
- PluginSet.java：プラグインの親クラス。
- GetArg.java：引数の値をタプル中から取得する為のクラス
- supersql.plugin：プラグイン用パッケージ
- supersql.plugin.plugins：プラグインを格納

4.2　プラグインの処理アルゴリズム

本章では、図 1 のサンプルクエリ 1 を用いて実際の処理の流れを交えて説明する。

4.2.1　処理の為の事前準備

まず、プラグインを処理するための事前準備について説明する。SuperSQL 処理系では、クエリを属性値を前から順に $0, 1, 2, \cdots n$(n は自然数) と数字に置き換えた形で扱われる。よって図 1 のクエリは

$$sch \ = \ [0, \ [1, \ 2, \ 3], \ 4]$$

といった形で扱われる。尚、0:fp.team, 1:fp.name, 2:fp.height, 3:fp.weight, 4:fp.height と対応している。よってデータベースから取得した段階、つまり PluginController クラスに渡される前の段階ではデータは以下のようなタプルとなる。

チーム	名前	身長	体重	身長

また、クエリをパーズする際、クエリ中に関数があった場合、$PluginList$ というリストを生成する。これは、クエリ中でどのような関数がどのポジションの属性値にかかっているかを格納しておくリストであり、関数名とポジションをペアで格納する。また、引数が複数の場合、ポジションを括弧で括る。サンプルクエリでは、2, 3 に BMI、4 に avg がかかってるので、$PluginList$ は以下のようになる。

$$PluginList \ = \ [(2 \ 3) \ BMI, \ 4 \ avg]$$

4.2.2　処理アルゴリズム

実際の処理アルゴリズムについて説明する。まず、dataconstructor 内の treegenerator クラスにおいて、クエリ中にプラグイン関数が存在した場合、データベースからクエリの結果として一旦取得した全てのタプル、sch、$PluginList$ を PluginController クラスに渡す。PluginController クラスでは、sch をレベルに分解し、上位レベルから再帰的に最下層レベルまでプラグイン関数の処理を行う。したがって、上記の sch は

$$sch1 \ = \ [0, \ 4]$$

$$sch2 \ = \ [1, \ 2, \ 3]$$

に分解され、$sch1$, $sch2$ の順に処理される。各レベル毎の処理は、先頭の属性値から $PluginList$ と照し合せながら見ていき、現在見ている属性値にプラグイン関数がかかっていない場合、グルーピングの基準値として保持しておき、レベルに関係なく追加していく。また、現在処理しているレベルが終了し、次のレベルに処理がうつる際に、現在のレベルの属性値を全て基準値に追加する。つまり、$sch1$ の場合、0 にはプラグインがかかっていないので基準値として保持するが、4 にはプラグインがかかっていないので基準値として保持しない。よって $sch1$ のプラグインを処理する際の基準値は

$$criteria\_set \ = \ [0]$$

となり、タプルは同じチームであるという基準によってグルーピングされる。そして、$sch1$ の処理が終わると $sch1$ の属性値を全て基準値として追加するので、基準値は

$$criteria\_set \ = \ [0, \ 4]$$

となる。次に、$sch2$ の場合、2,3 にはプラグインがかかっているので 1 のみが基準値として保持される。よってを $sch2$ のプラグインを処理する際の基準値は

$$criteria\_set \ = \ [0, \ 4, \ 1]$$

となり、チーム、チームの平均身長、名前がすべて同じであるという基準によってグルーピングされる。しかし、実際に処理を行う時、スカラー関数は各タプルごとに処理を施す関数であり、グルーピングという処理自体が必要ないため、集約関数であった場合のみ、$criteria_set$ に格納されている基準値によってタプルをグルーピングする処理を行い、グルーピングされたタプルをプラグイン関数に渡す。つまり、$sch1$ で処理される avg は集約関数であるので基準値 0 によってグルーピングされ、avg

に渡されるが、$sch2$ で処理される BMI はスカラー関数であるので、基準値 0,4,1 によらずグルーピングされずに BMI に渡される。この渡されるタプル (タプル集合) がプラグインで受け取る引数である $buffer$ である。また、もう 1 つの引数として $target$ というリストが渡されているが、これにはプラグインの引数になっている属性値のポジションが格納されている。第 1 引数のポジションから順に格納されており、BMI に渡される $target$ は

$$target \ = \ [2,\ 3]$$

となり、avg に渡される $target$ は

$$target \ = \ [4]$$

となる。そして、返ってきた結果を置き換える処理を行う。図 1 のサンプルクエリにおいて全てのプラグインの処理が終わった最終的なタプルは以下のようになる。

チーム	名前	BMI	BMI	平均身長

4.3 assign 関数を含む場合の処理アルゴリズム

ここでは図 2 のサンプルクエリを用いて説明する。

4.3.1 処理の為の事前準備

前章で説明した通り、プラグイン関数を処理する前の段階では以下のようになる。本システムでは assign 関数はプラグインとして処理している。また、本研究では、assign 関数に引数として与えられたプラグインは assign 関数内で自動的に呼び出し処理を行うため、PluginList には格納しない。

$$sch \ = \ [0,\ [1,\ 2,\ 3,\ 4],\ 5,\ 6]$$

$$PluginList \ = \ [(3\ 4)\ diff,\ (5,\ 6)\ assign]$$

ここで、ローカル変数保持用に $PositionList$ というリストを生成する。これはクエリ中で、どこにローカル変数が記述されているかを示すリストであり、このサンプルクエリでは以下のようになる。

$$PositionList \ = \ [4\ teamavg, 6\ teamavg]$$

また、$LPList$ というリストを用意しておく。これはローカル変数の代入用リストであり、ローカル変数名と、実際に参照すべきポジションのペアを格納しておくリストである。つまり、assign 関数が実行され、ローカル変数に値が代入されると更新されていく性質であり、初期段階では空である。

4.3.2 処理アルゴリズム

ここから実際の処理アルゴリズムについて述べていく。まず 4.2 章で説明したようにプラグイン処理の手順にそって進んでいく。つまり、このサンプルクエリにおいては上記の sch は

$$sch1 \ = \ [0,\ 5,\ 6]$$

$$sch2 \ = \ [1,\ 2,\ 3,\ 4]$$

に分解され、$sch1$, $sch2$ の順に処理される。ここでプラグイン側に処理を移す前に $PositionList$ を確認し、現在のレベルにローカル変数があればそのローカル変数名を用いて上記の $LPList$ を参照して代入する。$sch1$ においては 6 にローカル変数 $teamavg$ が存在しているが、$LPList$ を参照しても一致するローカル変数が存在しないため、assign 関数内でローカル変数名指定に用いられている記述であり、結果として代入は行われない。そして、$sch1$ で assign 関数が処理されると $LPList$ は以下のように更新される。

$$LPList \ = \ [teamavg\ 6]$$

次に $sch2$ では 4 にローカル変数 $teamavg$ が存在する。ここで $LPList$ を参照すると、ローカル変数 $teamavg$ は 6 の値を参照すればよいことが分かる。なぜなら assign 関数が実行された時点で 5,6 が 4 で参照したい avg の演算結果に書き換えられている為である。よって 4 に 6 の値を代入したのちプラグインに処理を移す。

表 1　整列処理の記述方法

種類	記述方法	使用例	意味
昇順	asc	(asc1)fp.height	身長の昇順に整列処理
降順	desc	(desc2)fp.name	選手名の降順に整列処理

5. 局所的整列

SuperSQL は 2 次元のフラットな表のみしか出力結果として得ることが出来ない従来の SQL とは異なり、ユーザの意図に応じて出力結果を自由に構造化することが可能である。従って、整列処理の機能には、ある基準値でグルーピングされたタプルの集合ごとに局所的に施せることが望まれる。よって本研究では、ある属性や演算結果に対して局所的に昇順もしくは降順に整列処理を施せることを可能にした。また、何段階にも構造化が可能であるので、例えば、チームによって整列処理を施した後、チーム内で身長によって整列処理を施したいという要望も出てくると考えられる為、加えて整列処理に順序という概念を取り入れた。

5.1 使 用 方 法

対象となる属性や演算結果の前に括弧を書き、その中に整列処理の種類や順序を記述する。その詳細を表 1 に示す。また、順序は数字を用いて、優先順位が高いものから 1 から順に指定する。

実際のクエリ中での使用例を図 2 のサンプルクエリ 2 に追記した形で示す。例えば、まずチームの平均身長によって昇順に並べ、次にチーム内において身長によって降順に並べた後、もし同じ身長の選手がいた場合、名前によって昇順に並べたい場合、図 6 のサンプルクエリ 3 のようになる。

6. システム評価

ここでは図 1、図 2、図 6 に示したサンプルクエリを実際に実行した結果を示す。尚、出力したい結果のコンセプトを変えないよう注意しながら各クエリに追加記述を施してある。各クエリの選手をポジション別にグルーピングする記述を施し、サ

```
1  /* sample3.ssql */
2  GENERATE HTML [
3      fp.team ,
4      [
5        (asc3)fp.name, (desc2)fp.height,
                                    diff(fp.height, teamavg)
6      ]! ,
7      (asc1)assign(avg(fp.height), teamavg)
8  ]!
9  FROM field_player fp
```

図 6　サンプルクエリ 3

Team	position	name	height	weight	BMI	avg (height)	avg (weight)
読売ジャイアンツ	内野手	小久保 裕紀	182	88	26.5668	180.625	87.25
		二岡 智宏	180	81	25.00		
		仁志 敏久	171	80	27.3588		
		清原 和博	188	104	29.4251		
	外野手	清水 隆行	183	83	24.7843		
		ローズ	182	87	26.2649		
		高橋 由伸	180	87	26.8519		
	捕手	阿部 慎之助	179	88	27.4648		
阪神タイガース	内野手	藤本 敦士	173	70	23.3887	178.875	80.50
		アリアス	180	93	28.7037		
		関本 健太郎	185	88	25.7122		
		今岡 誠	185	81	23.6669		
	外野手	金本 知憲	180	87	26.8519		
		桧山 進次郎	177	79	25.2163		
		赤星 憲広	170	65	22.4913		
	捕手	矢野 輝弘	181	81	24.7245		

図 7　実行結果 1-1

Team	position	name	height	diff(team avg)	diff(total avg)	team avg	total avg
読売ジャイアンツ	内野手	小久保 裕紀	182	△1.375	△2.25	180.625	179.75
		二岡 智宏	180	▼0.625	△0.25		
		仁志 敏久	171	▼9.625	▼8.75		
		清原 和博	188	△7.375	△8.25		
	外野手	清水 隆行	183	△2.375	△3.25		
		ローズ	182	△1.375	△2.25		
		高橋 由伸	180	▼0.625	△0.25		
	捕手	阿部 慎之助	179	▼1.625	▼0.75		
阪神タイガース	内野手	今岡 誠	185	△6.125	△5.25	178.875	
		藤本 敦士	173	▼5.875	▼6.75		
		アリアス	180	△1.125	△0.25		
		関本 健太郎	185	△6.125	△5.25		
	外野手	金本 知憲	180	△1.125	△0.25		
		桧山 進次郎	177	▼1.875	▼2.75		
		赤星 憲広	170	▼8.875	▼9.75		
	捕手	矢野 輝弘	181	△2.125	△1.25		

図 8　実行結果 2-2

ンプルクエリ 1 においてはさらに身長、体重の追加表示とさらに体重のチーム平均も求め表示している。また、サンプルクエリ 2 においては、チーム平均のみでなく全体平均も算出し、各選手毎にチーム平均との差に加え全体平均との差も表示している。また、図 8 の実行結果において、まず、チームの平均身長によって昇順に整列処理を施した後、チーム内で身長順に降順に整列処理を施し、もし同じ身長の選手がいた場合、昇順に整列処理を施した結果を図 9 として示す。

Team	position	name	height	diff(team avg)	diff(total avg)	team avg	total avg
阪神タイガース	内野手	今岡 誠	185	△6.125	△5.25	178.875	179.75
		関本 健太郎	185	△6.125	△5.25		
		アリアス	180	△1.125	△0.25		
		藤本 敦士	173	▼5.875	▼6.75		
	外野手	金本 知憲	180	△1.125	△0.25		
		桧山 進次郎	177	▼1.875	▼2.75		
		赤星 憲広	170	▼8.875	▼9.75		
	捕手	矢野 輝弘	181	△2.125	△1.25		
読売ジャイアンツ	内野手	清原 和博	188	△7.375	△8.25	180.625	
		小久保 裕紀	182	△1.375	△2.25		
		二岡 智宏	180	▼0.625	△0.25		
		仁志 敏久	171	▼9.625	▼8.75		
	外野手	清水 隆行	183	△2.375	△3.25		
		ローズ	182	△1.375	△2.25		
		高橋 由伸	180	▼0.625	△0.25		
	捕手	阿部 慎之助	179	▼1.625	▼0.75		

図 9　実行結果 3

6.1　実行結果に対する評価

結果を見てわかるように、従来の SuperSQL では実現不可能であった出力結果が得られている。また、図 8 に着目すると、*diff* による演算結果が正であれば△+(演算結果)、負であれば▼+(演算結果) という様に表示されているが、これは *diff* のプログラム中にそのような処理を追加するだけで実現している。このように関数という形で 1 段階抽象化することでユーザの意図によって、従来のクエリ記述方法では不可能であった細部の表現方法を実現可能にし、煩雑なクエリ記述を無くすことに成功している。当然これだけではなく、ユーザの意図によって自由に拡張可能であるので、様々なデータベースに対する多様なユーザの出力結果への表現要望を実現することが可能であると言える。

6.2　従来の拡張可能 RDBMS を用いる手法との比較

この様な機能拡張性は、POSTGRES [1] [2] において実現されているが、POSTGRES では出力結果が 2 次元であるのに対し、SuperSQL における出力結果は多次元であり、SuperSQL 処理系内部では構造化といった処理が必要となってくる。その為、処理系では一旦出力に必要なタプルを全て取得した後、クエリ記述を元に構造化していく段階で、必要に応じてその都度演算を行い演算結果を出力結果に組み込んでいく必要がある。よって、POSTGRES のユーザ定義関数を用いた手法は SuperSQL においては実現不可能であり、SuperSQL 処理系自体にプラグインとして機能を追加出来る本手法は有用性があると言える。

6.3　定義の自由度

演算系関数の定義については結果を見ても分かるように、またメディアに特化した関数については従来の SuperSQL の基本関数を全てプラグイン化することに成功したことから、ユーザからの要求を十分満たし得る結果となったと思う。しかし、assign 関数のように、基本的な内部処理方法を変えらざるを得ない関数についてはプラグイン関数のみで定義することは現在の段階では出来ない。しかし、内部仕様を変更しないと実現不可能な機能を追加する関数までプラグインで定義出来るようにしてしまうと結局内部仕様を知らないとプラグインが定義出

来ないため、内部仕様に手を加える場合とほとんど変わらなくなってしまい、内部仕様を知らなくても機能拡張を可能とするという本研究との主旨とずれてしまうため、その線引きについてはまだ検討の余地が残されている。

6.4 定義の難易度

まず、プラグインのプログラム記述に関して述べる。プログラム記述の際注意しなければならない規則は「戻り値の String 型指定」、「getArg メソッドの使用およびその戻り値の型」の 2 つである。これらは定式化されたものであり、なおかつ難解な規則では無いので、それに従ってプラグインのプログラムを記述することはそれほど難しくないと思われる。次にプラグインの定義記述に関して述べる。本研究において、ユーザが記述することが全部で 4 つあるが、それらは全て定式的なもので決して難しいものでは無い。しかし、プラグインの定義以外は自動的に生成出来る可能性があり、その部分を自動生成することが出来ればユーザの定義をさらに容易にすることが出来ると考えられるため、さらに検討の余地が残されている。

6.5 関数の入れ子のパターン

2.3 章で述べたように、「下位レベルの演算結果を上位レベルで用いたい」ことを目的に集約関数の引数に集約関数を与える場合のみ、本研究では不可能である。なぜなら、本研究においては、上位レベルから下位レベルへ向けて処理を行って行くため、すでに参照したいプラグインの処理が終わってる「上位レベルの演算結果を下位レベルで用いたい」場合は assign 関数を導入することによって解決されたが、この場合はまだ処理がされていない演算結果を先取りする形になるためである。つまり、これを実現するためには外側のプラグインで受け取ったタプルをクエリの記述に基づいて、ある基準値に基づいてさらにグルーピングして演算を行う関数を呼び出し引数のプラグインの演算結果を得なければならないのだが、上位レベルのプラグインを処理している段階で下位レベルのグルーピングの基準値を知ることは不可能であるためである。引数がスカラー関数であれば実現可能な理由は、引数のプラグインの演算結果を求めるときにタプルをグルーピングする必要が無いためである。
この解決方法の案としては、

（1） このパターンのみ何か別の処理方法を行うようにする方法

（2） クエリ構造を分析しながらプラグインを実行するのではなく、一旦クエリ構造を全て分析し、レベル別にグルーピングの基準値などを全て保持した上で上位プラグインから順に処理を行う方法

（3） 関数の入れ子は使用不可能にし、他の演算結果を参照したい場合はすべてローカル変数を用いることにし、プラグインを処理する場合、レベルに関係なく、assign 関数を優先的に一旦すべて処理した後に他のプラグインの処理にうつる方法

などが挙げられる。これらの案を含め、今後検討して行かなければならない事項である。

6.6 ローカル変数の使用範囲

ローカル変数を使用する場合、SuperSQL においては

- 同一クエリ内で使用する場合
- 複数のクエリに渡って使用する場合

の 2 パターンが考えられる。そこで、ここでは各パターンについてその使用範囲を検討する。

6.6.1 同一クエリ内で使用する場合

本研究において、ローカル変数は上位レベルの演算結果を下位レベルで使用することを前提としている為、下位レベルでの演算結果をローカル変数に代入して上位レベルで使用することは不可能である。この理由については 6.5 章で述べた通りである。しかし、レベルの処理順序をユーザが意識せずに自由にローカル変数を定義、使用出来るようにすることは非常に有用であり、かつクエリ記述が容易になると考えられる。この解決方法としては、6.5 章で述べた解決方法案の 2,3 を組み合わせた案が一番有力であると考えている。この事項については今後、さらに検討し、実現していきたいと考えている。

6.6.2 複数のクエリに渡って使用する場合

SuperSQL 処理系では複数のクエリから結果出力を構成することが出来る [5]。そのため、あるクエリで定義したローカル変数を他のクエリで使用出来るようになることが望ましい。しかし、本研究でのアプローチは、すでに演算処理が行われて演算結果に書き換えられた属性値を用いてローカル変数の参照を行う。つまり、複数のクエリから出力結果を得ようとした場合、例えば 1 つ目のクエリで assign 関数を用いてチーム平均値を *avg* というローカル変数に代入し、*avg* を用いてチーム平均値を 2 つ目のクエリで使用しようとした場合、2 つ目のクエリにおいては当然チーム平均値を求める為のプラグイン関数は実行されていないので、参照したい値、つまりチーム平均値は自らのタプル内には存在せず、結果として参照出来ない。つまり、本研究のアプローチは 1 つのクエリ内で使用することを前提としている為、複数のクエリに渡ってローカル変数を使用することは現段階では不可能である。よって今後の課題であると言える。

7. おわりに

本研究では、SuperSQL 処理系におけるプラグイン関数および局所的整列を提案し、内部使用に手を加えることなく、プラグインという形で拡張モジュールを用意するだけでユーザが独自に機能拡張出来ることを可能にした。

文　献

[1] Michael Stonebraker, Lawrence A. Rowe, “The Design of Postgres”, *SIGMOD Conference 1986*, pp. 340-355, 1986

[2] Michael Stonebraker, Jeff Anton, Michael Hirohama, “Extendability in POSTGRES”.*IEEE Data Eng. Bull. 10(2)*, pp. 16-23 (1987)

[3] 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

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

[5] 石川恭子, 有澤達也, 遠山元道, “データ集約型 Web サイトにおける静的生成コンテンツの部分更新”, 情報処理学会論文誌 *IPSJ-TOD4613002*, 2005