SuperSQL 処理系におけるプラグイン関数および局所的な整列の実現

Implementation of Plug-in Function and Local Sorting in SuperSQL

中道 亮♦ 金 翰基♦
遠山 元道♠
Ryo NAKAMICHI Hanki KIM
Motomichi TOYAMA

SuperSQL 処理系では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，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為，異なるレベル間での値の参照を可能にする assign 関数を実装し，SuperSQL 処理系に代入という新しい概念を導入した．最後に，本研究ではグルーピングされたタプル集合内での局所的な整列も実現した．

Extendability of SuperSQL for fulfilling various demands of users is important. Our approach is that avoid change of the language specification accompanying extension as much as possible and users do not have to modify original code of SuperSQL. Therefore, in this paper, we propose a plug-in mechanism, which has expandability to SuperSQL so 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 propose assign function, which makes refer to value between different levels so that we could introduce the new concept of substitution into SuperSQL. Finally, we propose local sorting to the grouped taple set.

1. はじめに

SuperSQL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る．しかし，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算術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必要がある．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が，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することを可能とした．プラグイン関数開発の自由度を向上させるため，あるネストレベルのプラグイン関数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．

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. SuperSQL とは

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．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，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．

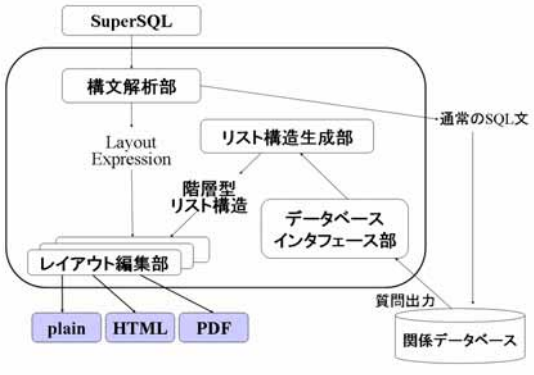

図 1 SuperSQL 処理系
Fig.1 SuperSQL System Configuration

3. 本システムの概要

3.1 プラグイン関数機構

本研究では，ユーザが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関数の開発とその仕様定義のみである．ユーザは開発したプラグイン関数を所定のパッケージ内

♦ 学生会員 慶應義塾大学大学院理工学研究科修士課程
{ryo, hanki }@db.ics.keio.ac.jp
♠ 正会員 慶應義塾大学理工学部情報工学科
toyama@ics.keio.ac.jp

に置き，仕様定義を所定の仕様定義専用ファイルに記述すれば，プラグイン関数を用いたクエリを記述，実行可能である．また，プラグイン関数が処理されると，引数として与えられた属性値が演算結果に書き換えられる．例えば，「平均身長を求める」集約演算を行うため，身長を引数に取るavg関数を定義したとする．この場合，avg関数が処理され平均身長が求まると，平均身長を求める為に使用したタプル集合の各タプルの引数として与えられた属性「身長」の値が平均身長に書き換えられる．

ここで，スカラー演算であるBMI値などであれば，クエリ中に演算子が記述できればこの様にプラグイン関数を定義する必要が無いように感じられる．しかし，プラグイン関数として定義することで，BMI値が基準値を越えている人に対してユーザが警告の意味でBMI値の前に「！」を表示させたい要望を持っているとすれば，プラグイン関数のプログラム中に記述すれば容易に実現可能であるが，演算子を用いる方法ではこの様な表現方法の要望は不可能であり，プラグイン関数の利点の1つといえる．

3.2 assign 関数の導入

プラグイン関数を定義するにあたって，あるプラグイン関数によって処理された演算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考えられる．本研究ではプラグイン関数をネストの上位レベルから順に処理していく処理アルゴリズムを用いている為，下位レベルのプラグイン関数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関数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い．そこで，上位レベルのプラグイン関数を処理して得られた演算結果を一旦ローカル変数として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，値そのものをローカル変数として保持する方法が挙げられる．しかし，SuperSQLの特性上，あるネストレベルでの演算結果は複数存在し得るので，同一のローカル変数名で保持出来ない為不可能である．よって，本研究ではプラグイン関数が処理された時点でタプル内の引数として与えられた属性値が演算結果に書き換えられるという点を生かし，自らのタプル内の値を参照するという方法を採用し，独自にassign関数と呼ぶ関数を導入することによって値の参照を実現した．

3.3 assign関数の定義と使用方法

assign関数の定義は以下のようになる．

- assign(<式>[1], <ローカル変数名>)

第1引数には下位レベルで参照したい値を返す式，第2引数にはローカル変数名を記述する．例えばユーザが

- assign(avg(引数, … , 引数), teamavg)

と記述すれば，処理系内部ではassign関数が記述されているネストレベルでavg関数が実行され，タプル内での演算結果があるポジションとローカル変数のペアリストが作成される．

下位レベルで参照する際は，ユーザは参照したい演算結果に対応するローカル変数をプラグイン関数の引数に与えるだけでよい．処理系内部ではクエリのパーズ中にローカル変数を発見すると，そのローカル変数が与えられているプラグイン関数処理の前段階でペアリストへの問合せが行われ，引数として与えられたローカル変数が参照すべきポジションの値がローカル変数のポジションへ代入される．そして代入された値を用いてプラグイン関数を実行する．この方法であれば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関数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．

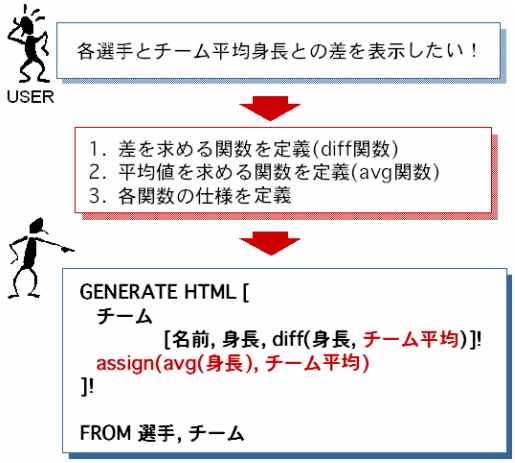

図2 ユーザのワークフロー

Fig.2 User's Workflow

4. プラグイン関数の定義方法

4.1 プラグイン関数定義用テンプレート

プラグイン関数定義用のテンプレートファイルを以下に示す．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関数を定義する．基本的にユーザは処理記述部分に要望の処理を行うプログラムを記述すればよい．また，必要なクラスやパッケージ等があれば自由に追加可能であり，このテンプレートを用いることによって，内部処理系に手を加えずに機能を追加することが可能である．

```
package supersql.plugin.plugins;

import supersql.extendclass.ExtList;
import supersql.plugin.PluginSet;
import supersql.plugin.GetArg;

public class template extends PluginSet{

    public String exe(ExtList buffer, ExtList target){

/*
 *
 *            ユーザ定義関数の記述
 *
 */


            //結果は String 型で
            String result = "result";

            return result;
    }

}
```

図3 プラグイン定義用テンプレート

Fig.3 Template file for Plug-in

しかしプログラム記述の際，「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」という2つの記述規則が存在する．前者のプラグイン関数の戻り値はString型でなければならない理由は，クエリ中で使用されているプラグイン関数のインスタンスを処理系内部で自動的

[1] 現在，プラグイン関数および属性の記述が可能である

に生成し，実行するためである．後者のgetArgメソッドとは，SuperSQLの知識が乏しいユーザでもプラグイン関数を開発可能にするために用意したものであり，クエリ中で引数としてプラグイン関数に与えられた属性値をタプル内から取得するためのメソッドである．このメソッドの使い方を覚えておけば，SuperSQLの知識が無くとも演算処理のプログラムさえ記述できれば，プラグイン関数を開発することが可能である．また，このメソッドの戻り値はString型であるためプログラム記述の際は，必要に応じて型を変換する必要がある．

以上の様に記述規則が存在するが，これらの規則は定式化されており，かつ難解では無いので，規則に従って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．

4.2 プラグインの定義例

以下に実際のプラグイン関数の定義例を示す．ここでは図2で用いたDiff関数の定義を用いる．

```
package supersql.plugin.plugins;

import supersql.extendclass.ExtList;
import supersql.plugin.GetArg;
import supersql.plugin.PluginSet;
import java.text.NumberFormat;

public class Diff extends PluginSet{

    public String exe(ExtList buffer, ExtList target){

        double height = Double.parseDouble( GetArg.getArg(buffer, target, 0) );
        double avg = Double.parseDouble( GetArg.getArg(buffer, target, 1) );

        double diff = height - avg;

        NumberFormat formatter = NumberFormat.getNumberInstance();
        formatter.setMaximumFractionDigits(4);
        formatter.setMinimumFractionDigits(2);

        String tmp = formatter.format(diff);
        StringBuffer resultbuffer = new StringBuffer();

        if(diff >= 0){
            resultbuffer.append("△" + tmp);
        }else{
            resultbuffer.append("▼" + tmp.toString().substring(1));
        }

        String result = new String(resultbuffer);
        return result;

    }

}
```

図 4 diff 関数の定義例
Fig.4 A Example of Diff Fucntion

ここで特徴的な点は表現方法を変更する為の記述の部分であり，3.1章で述べたユーザの表現要望に応じた処理を行う為の記述である．この例の記述は演算結果が正であれば△+(演算結果)，負であれば▼+(演算結果)という様に表示している．

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，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整列処理を施すことが望まれる．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すことを可能にした．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ある為，整列処理に優先順序の指定が出来る．

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．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．その詳細を表 1 に示す．

表 1 整列処理の記述
Table 1 The Description Method of Alignment Processing

種類	記述方法	使用例	意味
昇順	Asc	(asc+数字)属性	属性値の昇順に整列
降順	Desc	(desc+数字)属性	属性値の降順に整列

また，実際のクエリ中での使用例を図 2 のクエリ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 5 のようなクエリにようになる．

```
GENERATE HTML [
    チーム,
        [ (asc3)名前, (desc2)身長, diff(身長, チーム平均) ]! ,
 (asc1)assign(avg(身長), チーム平均)
]!
FROM 選手, チーム
```

図 5 局所的整列を用いたクエリ例
Fig.5 A Sample Query using Local Alignment

6 評価・検討

6.1 実行結果

ここでは図 2，図 5 に示したクエリに，見やすさの為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若干の追加記述を施したクエリを実行した結果を図 6，図 7 として示す．以下，追加記述した内容を示す．各クエリにおいて選手をポジション別にグルーピングし，図 2 のクエリにおいては，チーム平均のみでなく全体平均も算出し，各選手ごと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 6 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，名前の昇順に整列処理を施した結果が図 7 である．

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 6 実行結果 1
Fig.6 A Result 1

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 7 実行結果 2
Fig.7 A Result 2

6.2 実行結果に対する評価

結果を見てわかるように，SuperSQL の特性に合った整列結果が得られている．また，Diff 関数は 4.2 章で示したものを用いている．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あった細部の表現方法を実現し，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，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．

6.3 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[3][4]において実現されているが，POSTGRES では出力結果がフラットな関係であるのに対して，SuperSQL における出力結果はネストされた表であり，処理系内部では構造化処理が必要である．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処理を行い，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 に直接適応することは不可能である．

6.4 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．

(ⅰ)同一クエリ内で使用する場合

本論文中では，ネストの下位レベルから上位レベルのローカル変数を使用する場合を用いて説明したが，ネストの上位レベルから下位レベルのローカル変数を使用する場合については現段階では実現していない．3.2 章でも述べた様に本研究ではプラグイン関数をネストの上位レベルから順に処理していく処理アルゴリズムを用いている為，まだ処理されていない演算結果を先取りする形になるためであり，assign 関数の仕様を含め現在検討中である．

また，現段階では GENERATE 句内での使用を前提としているが，他の句でも使用可能にすることによって新たな可能性が広がることも考えられる．

(ⅱ)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の値を用いてローカル変数を介した値の参照を行う為，例えばクエリ 1 で定義したローカル変数をクエリ 2 で使用しようとした場合，クエリ 2 においてはローカル変数として定義したプラグイン関数は実行されていないので参照したい値はクエリ 2 で取得したタプル内には存在せず，結果として値の参照は出来ない．この様に，現段階では複数のクエリに渡ってローカル変数を使用することは不可能であり，今後の課題であると言える．

7. まとめ

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，実装し，内部使用に手を加えることなく，プラグイン関数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，assign 関数を導入することによって，SuperSQL 処理系に代入という新しい概念を導入することが出来た．また，グルーピングされたタプル集合内に対する局所的な整列処理も可能となった．

今後の課題としては，プラグイン関数の定義の難易度，自由度の改善や，ローカル変数の使用範囲の拡大などがあげられる．

[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] Michael Stonebraker, Lawrence A. Rowe.: “The Design of Postgres”, SIGMOD Conference 1986, pp. 340-355. (1986)
[4] Michael Stonebraker, Jeff Anton, Michael Hirohama.: “Extendability in POSTGRES'”, IEEE Data Eng. Bull. 10(2), pp. 16-23 (1987)
[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)

中道 亮 Ryo NAKAMICHI
慶應義塾大学大学院理工学研究科修士課程在学中．2006 慶應義塾大学理工学部情報工学科卒業．データベースの研究に従事．日本データベース学会学生会員．

金 翰基 Hanki KIM
慶應義塾大学大学院理工学研究科修士課程在学中．データベースの研究に従事．日本データベース学会学生会員．

遠山 元道 Motomichi TOYAMA
慶應義塾大学理工学部情報工学科専任講師．博士（工学）．1984 慶應義塾大学大学院博士課程単位取得退学．主にデータベースシステムの研究に従事．IEEE Computer Society, ACM, 情報処理学会，日本ソフトウェア科学会，電子情報通信学会，日本データベース学会正会員．