SuperSQL 構文解析部の再構築によるエラーメッセージの高品質化

田嶋　将大† 　五嶋　研人† 　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: †{taji,goto}@db.ics.keio.ac.jp, ††toyama@ics.keio.ac.jp

あらまし　SuperSQL は，独自のクエリを記述することによって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 の拡張言語である．従来の SuperSQL の研究開発において，その機能の拡張などの実装が行われ，必要に応じて構文解析部の構築や修正が行われてきた．このことから考えられる問題として，エラー処理が適切に行われず，ユーザーへのエラー表示が適切に提示されないことなどがある．そこで，本論文では SuperSQL クエリの構文解析ルールの定義を行うことで構文解析部を再構築することで，エラー処理品質の向上させ，ユーザーに対するエラーメッセージをより正確にすることを目指した．

キーワード　SuperSQL，SQL，構文解析，パーサー，ANTLR

1. はじめに

SuperSQL は，独自のクエリを記述することによって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 の拡張言語である．通常の SQL では，シンプルでフラットな表しか再現できないが，SuperSQL を用いることで様々な表を作成することができる．また，SuperSQL を用いると，HTML と PHP などを用いたサーバサイトプログラミングなどによる一般的な方法に比べて，はるかに少ない行数で Web ページを生成することができる．

従来の SuperSQL の研究開発は，その機能の拡張などの実装が行われてきて，その度に必要に応じて構文解析部の構築や修正が行われてきた．このことから派生する問題点として，SuperSQL 特有のクエリの複雑な構文解析ルールが不明瞭になってきてしまっていること，エラー処理が適切に行われず，ユーザーへのエラー表示が適切に提示されないことがあることなどが考えられる．そこで，本研究では SuperSQL クエリの構文解析ルールの定義を行うことで構文解析部を再構築し SuperSQL の構文解析のルールを明確にし，今後の開発に役立てることを目的とし，またそれによって，エラー処理の品質を向上させ，ユーザーに対するエラーメッセージをより正確にすることを目指した．

2. SuperSQL とは

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，Mobile_HTML5 [3] 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 SuperSQL アーキテクチャ

SuperSQL のアーキテクチャを図 1 に示すん．SuperSQL 処理系は，構文解析部 (Parser)，リスト構造生成部 (List Constructor)，メディア生成部 (Code Generators) から成る．SuperSQL クエリが発行されると，まず構文解析部で通常の SQL 文とレイアウト式に分け，SQL 文を DBMS に渡して検索結果を受け取る．リスト構造生成部では受け取ったフラットな結果に対し，レイアウト式に従って階層的な構造を持たせる．最後にメディア生成部がクエリで指定したメディアファイルを結果として出力する．

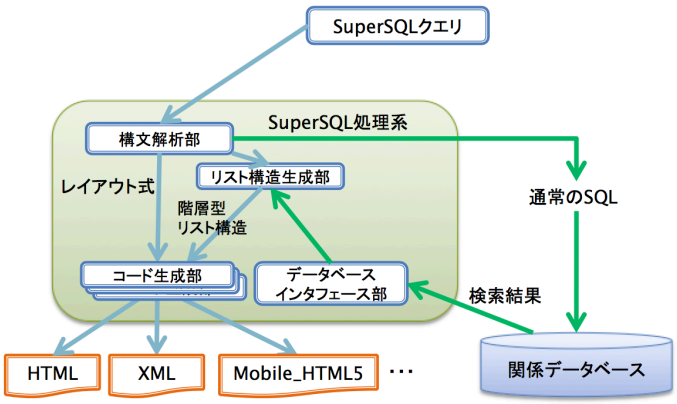

図 1　SuperSQL アーキテクチャ

2.2 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する．

例：name, place

name	place

- 垂直結合子 (!)

データを縦に結合して出力する．

例：name! place

name
place

- 深度結合子 (%)

データを 3 次元方向へ結合する．出力が HTML ならばリンクとなる．

例：name % place　　name → place

2.3 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返して表示する．また反復子はただ構造を指定するだけではなく，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

[部署名]! , [雇用者名]! , [給料]!

とした場合には各属性間に関係はなく，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

[部署名! [雇用者名, 給料]!]!

とした場合には，その部署毎に雇用者名と給料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

2.4 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．

[name@{width=100, color=red}]!

2.5 関　　数

2.5.1 image 関数

image 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．

image(pict, “./pic”)

2.5.2 link 関数 (出力メディアが HTML の場合のみ)

link 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．

link(name, “./menu.ssql”, place)

3. SuperSQL 構文解析部の再構築

この章では，実際のクエリを例にして本研究で行った内容について述べる．

以下クエリ 1 では，image() 関数内の赤文字になっている” , ”が余分であるために，誤ったクエリとして認識されるべきである．

```
<クエリ 1 >
GENERATE Mobile_HTML5{
        header( ”映画レビュー一覧” )!
         [image(m.image , ),
            {
             {m.title}@{bgcolor=BlanchedAlmond}!
             line()!
            {’ 公開年：’ ||m.year!
             line()!
            ’ 制作会社：’!
            c.name !
            line()!
            ’ 映画ジャンル：’!
            g.name}
            }@{width=150}
        ]!5%@{ dynamic}
}
FROM movie m, company c, genre g
WHERE m.company = c.id and m.genre = g.id
```

3.1 従来 SuperSQL 構文解析部による問題点

図 2 は従来の SuperSQL の構文解析部によるシンタックスエラーメッセージの例であり，クエリエラーの内容に依って実際のエラー要因と指摘されるエラーメッセージとがずれてしまうことが問題として挙げられる．図 2 の例では，実際のエラー箇所は image 関数内の記述であるのに対して，指摘されるエラー要因はその後の ”] ”を指してしまっている．この差によって，ユーザーは実際にエラーが生じているところに着目できず，クエリ訂正までにかなりの時間を費やしてしまうことがある．

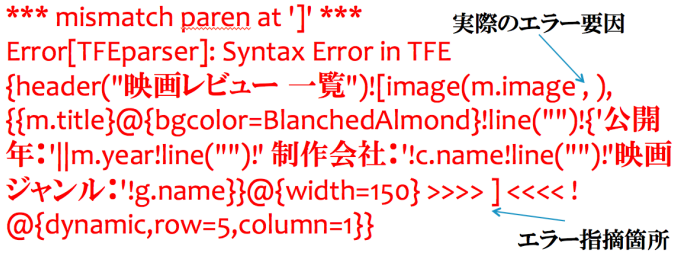

図 2　従来の構文解析によるエラーメッセージ

3.2　ANTLR

本研究では，構文解析部の構築に，LL(*) 方式 の解析手法をとる構文解析部を生成する ANTLR [4] というパーサージェネレーターを使用した．一般に用いられる構文解析手法には，LL 方式と LR 方式があるが，LR 方式ではエラーの検出は早いことが言えるが，LL 法に比べてコンパイルエラー時のエラーメッセージを適切に出しにくい という欠点があり，本研究の目的の一つである，ユーザーへのエラーメッセージの正確さの向上の実現の為に LL 方式の構文解析手法を採用した．図 3 のように SuperSQL クエリの記述ルールの定義を行い，SuperSQL クエリの構文解析ファイルや字句解析 ファイルを生成させている．また，これによって，構文解析手法を明確にすることができ，その確認が容易になると考えられる．

```
media   :   (K_GENERATE | error) IDENTIFIER  ;

operand    :
            (
                (function
                | OPEN_BRACE{braceflag++;} exp CLOSE_BRACE{braceflag--;}
                | (sorting)?((table_alias '.')? column_name)
                | grouper
                | composite_iterator
                | if_then_else
                | STRING_LITERAL
                | NUMERIC_LITERAL
                | keyword
                )
                ('||' operand)*
            )
            (DECOLATOR)?
            ;


grouper :
        OPEN_BRACKET{bracketflag++;}
        exp
        CLOSE_BRACKET{bracketflag--;}
        C1
    |
        OPEN_BRACKET{bracketflag++;}
        exp
        CLOSE_BRACKET{bracketflag--;}
        C2
    |
        OPEN_BRACKET{bracketflag++;}
        exp
        CLOSE_BRACKET{bracketflag--;}
        C3
    ;
```

図 3　SuperSQL 構文解析ルール定義 (一部抜粋)

3.3　SuperSQL 新構文解析部による解析の流れ

また，SuperSQL クエリには，下記のクエリ 2 のように foreach 句や Session 関数などのような GENERATE 句の前に記述される特殊な記述が存在する．

```
<クエリ 2 >
FOREACH p.team, p.num
GENERATE HTML{
 [{p.name, link(t.name, "team.ssql", t.id)}!
image(p.pict, "photo")]!
}
FROM player p, team t
WHERE p.team = t.id
```

これらの記述も含めて構文解析ルールを定義するにあたり，これらの記述の中にはその後のクエリ (GENERATE 句以降) に対して処理を行うものも存在するため，それらの処理に対応するために，SuperSQL クエリの構文解析を図 4 のような流れで行うよう設計した．まず，GENERATE 句前後でクエリの切り分けを行い，GENERATE 句前に記述があれば，その記述ように定義された構文解析ファイルによって解析を行い，その記述内容によって必要であれば GENERATE 句以降のクエリに対しての処理を行い，その後，GENERATE 句以降の構文解析を行う．

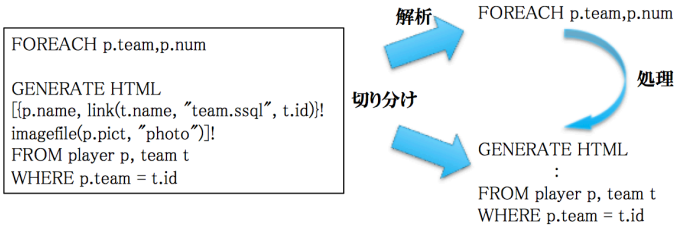

図 4　SuperSQL 構文解析の流れ

3.4　エラー処理とエラーメッセージ

ここで，この生成されるプログラムによるクエリの構文解析中で定義したルールに反する記述が発見された時，その例外が発見された時のクエリ中のトークンを保持し，そのトークンに基づいてエラー処理を行う処理系が ANTLR の内部で定義されている．この処理系によって SuperSQL クエリのエラーメッセージを生成させると，図 5 のようになる．このようなエラーメッセージでは，視覚的に実際のエラー箇所が何処であるのかを特定するには不十分であると考えられ，クエリが長くなれば，その特定にはかなりの時間とユーザーの負担の大きさが見込まれてしまう．

```
line 3:16 no viable alternative at input
'{\n \theader( "映画レビュー一覧" )! \n \t[image(m.image,)'
```

図 5　ANTLR 内部処理によって生成されるエラーメッセージ例

そこで，本研究では，SuperSQL クエリ特有のエラーに合わせてエラー処理を行うように，またそのエラーパターンに対応させる形で，ユーザーにエラーメッセージを適切かつ簡潔に提示できるように，この内部処理系と別で定義を行っている．本研究で定義された処理系では，ANTLR の内部処理系によって保持されるエラー要因となっているトークン，その前後のトークン

や記述内容によってエラー要因を推定し，そのエラー箇所の明示とエラーメッセージの生成を行っている．エラー要因の判定のフロー図を図 8 に示す．

例えば，クエリ 3 のようなクエリの場合，保持されるエラートークンは”FROM”になり，そのエラートークンの一つ前のトークンが’] ’ であり，2.2 で取り扱った反復子の記述において，反復子に必要な’ , ’ や’ ! ’ の記述が抜けてしまっていると判定を行う．

```
<クエリ 3>
GENERATE HTML{
 [ e.id, e.name! e.salary ]
}
FROM employee e
```

このエラー判定によるエラーメッセージは図 6 のようになる．

```
line 3:0 parse error detected.
The OffendingToken is : '}'
GENERATE HTML{
[ e.id, e.name! e.salary  >>>>>]<<<<< }
FROM employee e
*****did you mean ']!' or '],' ?*****
```

図 6　反復子エラーメッセージ例

また，2 章では　取り上げなかったが，SuperSQL では単一属性の前に (asc)，(desc) を記述することでその属性でのソーティングを可能にする．これに関して，クエリ 4 では，エラートークンは”s.name”の”s”が保持され，そのエラートークンの直前に asc，desc の記述の有無の判定を行い，その記述が無い為，次に関数の記述が存在するかの判定を行う．

```
<クエリ 4>
GENERATE HTML
{
 [(act1)s.name@{width=50},
(acct2)d.name||'F'@{width=30},
(act3)d.name@{width=130, bgcolor=yellow},
m.name@{width=100, bgcolor=azure},
i.name@{width=120}]!@{bgcolor=beige}
}@{cssfile=winter.css}
```

クエリ 4 では asc，desc の記述が無く，関数の記述も存在しない為，ソーティングの記述が誤っていると予想される．このエラー判定によってソーティングのエラーメッセージは図 7 のようになる．

```
NoViableAlt
line 3:8 parse error detected.
The OffendingToken is : 's'
generate html
{
        [ >>>>>(act1)<<<<< s.city@{width=50},
        (act2)d.floor||'F'@{width=30},
        (act3)d.name@{width=130, bgcolor=yellow},
        m.name@{width=100, bgcolor=azure},
        i.name@{width=120}]!@{bgcolor=beige}
}@{cssfile=winter.css}

*****did you mean '(asc)' or '(desc)' or 'IF function:( )?' ?*****
```

図 7　ソーティングエラーメッセージ例

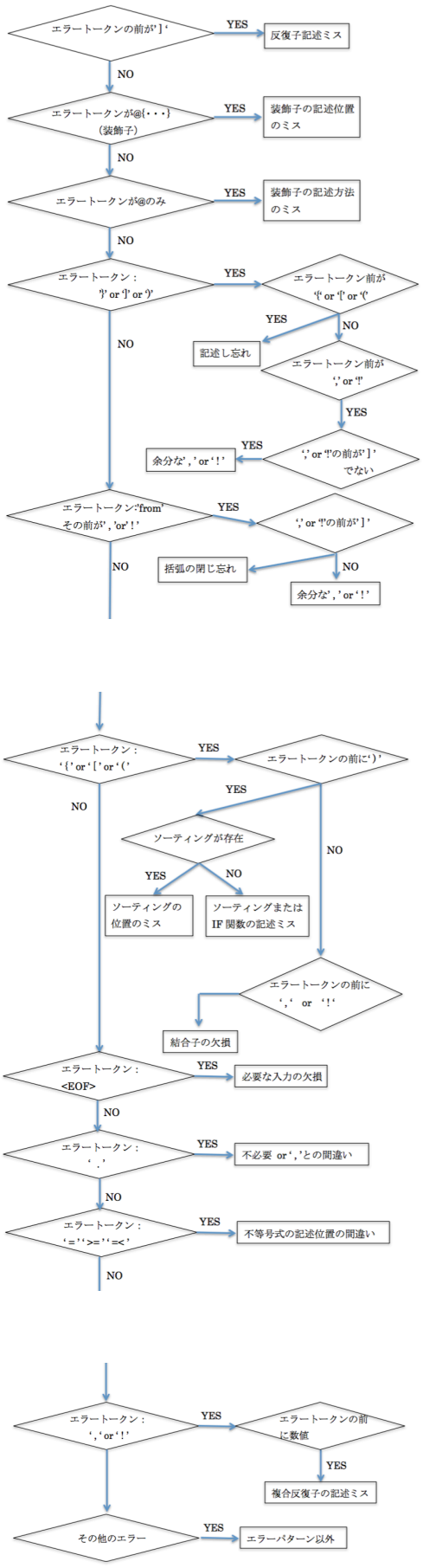

図 8　エラー処理判定フロー図

この処理によって，上記のクエリ 1 の構文解析を行った結果のエラーメッセージが図 9 で ある．これによって，エラー要因のクエリ中での位置の特定が，視覚的に行えるようになったため，エラー箇所の特定に掛かる時間の削減が見込まれるようになったとともに，実際のエラーの原因と考えられる箇所を従来のエラーメッセ ージと比べてより正確に指摘することができるようになった．

```
line 3:16 parse error detected.
The OffendingToken is : ')'
GENERATE Mobile_HTML5 {
        header( "映画レビュー一覧" )!
        [image(m.image >>>>>,<<<<< ),
        {
                {m.title}@{bgcolor=BlanchedAlmond}!
                line()!
                {'公開年：' ||m.year!
                line()!'
                制作会社：'!
                c.name !
                line()!
                '映画ジャンル：'!
                g.name}
        }@{width=150}]!5%@{ dynamic}
}
FROM movie m, company c, genre g
WHERE m.company = c.id and m.genre = g.id

*****extra ',' detected before ')'*****
```

図 9　新構文解析部によるエラーメッセージ

4. 評　　価

本研究の評価のため，慶應義塾大学理工学部情報工学科の 3 年生を対象とした授業であるデー タモデリングの SuperSQL を用いた最終課題にお いて，実際に 3 年生の記述したクエリのログを用いて，それぞれの構文解析を行う．そのクエリログには全 9862 クエリが採取されており，それぞれのクエリに対して SuperSQL の実行が成功したか失敗したかが情報として保持されている．この情報を用いて，これらのクエリの中から，実行の失敗となったクエリ (全 2373 クエリ) のみを収集し，それぞれのクエリのエラー原因の特定とその分類を行った．その分類集計の結果を図 10 に示す．

図 10 に分類されたクエリ群に対して，従来の SuperSQL の構文解析部のエラー処理と本研究によるエラー処理がそれぞれどれだけ正確にエラー要因の推定とエラー箇所の特定を行えているのかの検証を行った．その結果を表 1 に示す．この表は，エラーパターンそれぞれに対して，旧エラーメッセージが正確にエラー箇所の指摘とエラー原因を特定できていたかの割合と，本研究によるエラーメッセージが正確にエラー箇所の指摘とエラー原因を特定できていたかの割合を示しており，エラーパターンの頻出度が高い人順に表示されている．

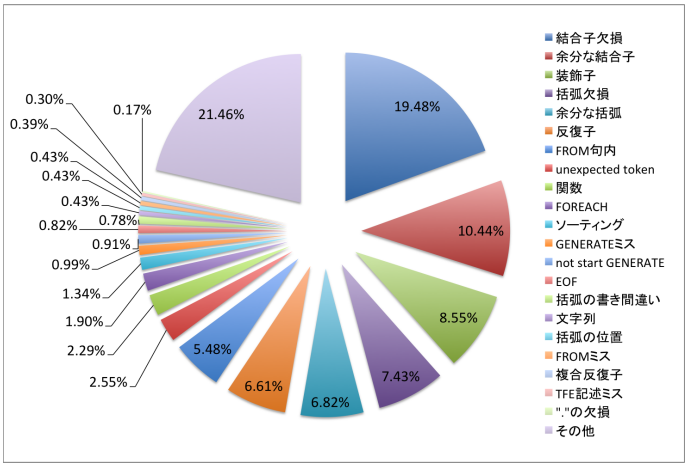

図 10　SuperSQL クエリエラーパターン分類・集計

表 1　SuperSQL クエリエラーパターンとその対応の比較

エラーパターン	エラー検出精度 (旧エラーメッセージ)	エラー検出精度 (新エラーメッセージ)
結合子欠損	94.9%	99.6%
余分な結合子	79.3%	99.2%
装飾子	73.5%	94.9%
括弧欠損	46.5%	80.6%
余分な括弧	45.6%	76.4%
反復子	93.5%	96.7%
FROM 句内	0%	88.5%
unexpected token	45.8%	81.4%
関数	41.5%	56.6%
FOREACH ミス	100%	0%
ソーティング	30%	74.2%
GENERATE ミス	100%	100%
not start GENERATE	100%	76.2%
EOF	0%	100%
括弧の書き間違い	94.4%	88.9%
文字列	80%	60%
括弧の位置	70%	50%
複合反復子	55.5%	66.7%
TFE 記述なし	0%	100%
“ . ”の欠損	94.4%	88.9%
その他	-	-

この表から SuperSQL のクエリによく見られる，上位 8 個のパターンのエラーに対しては，従来のエラーメッセージよりもより正確にエラー箇所の指摘およびエラー原因の特定を行えるようになったことがわかる．

エラーパターンの割合は小さいが，TFE 内の必要な記述の欠

如や括弧内への記述の欠如に関しては，従来では，そのエラーの検出はできていなかったが，新構文解析部ではこのようなエラーへの対応ができるようになった．

また，精度が下がってしまったものとして，FOREACH 句の記述ミスに関しては，従来の構文解析部では，GENERATE 句前に”FOREACH”などの特定の関数名の有無を判定することを行っていたが，本研究における構文解析部では，GENERATE 句前の記述の方法を定義したために，特定の関数の記述ミスなどが検出できないということが原因として考えられる．また，not start GENERATE には，SuperSQL クエリ以外の記述や GENERATE 句の前に TFE の記述がされているようなクエリが分類されているが，FOREACH 句の記述ミスの場合と同様に GENERATE 句の前に特定の関数の記述以外が存在する場合には GENERATE 句でクエリを始めるように促すエラーメッセージを提示する．これに対し新構文解析部では，記述がルールに一致するか否かの判定のみを行っているために，GENERATE 句が含まれているクエリでは，旧エラーメッセージと同様なエラーメッセージの提示を実現することができなかった．

参 考 文 献

[1] SuperSQL: http://SuperSQL.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] 五嶋 研人, 遠山 元道. “ SuperSQL によるモバイル Web アプリケーション生成機構の実装 ”, 慶應義塾大学　修士論文, 2013.

[4] Terence Parr: “Adaptive LL(*) Parsing: The Power of Dynamic Analysis”, Proceedings of ACM International Conference on Object Oriented Programming Systems Languages & Applications, pp. 579-598, 2014