SuperSQLにおける窓関数機構の実装

廣瀬　桂大† 　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: †{harry,goto}@db.ics.keio.ac.jp, ††toyama@ics.keio.ac.jp

あらまし　データをデータベースから取り出す際に, 窓関数を用いる事により, 複雑なクエリを用いる事なくそのデータを活用する事が可能となる. 本研究では, 代表的な窓関数を SuperSQL 上に実装し, 通常の SQL の文法と比べて, 汎用性を損なう事なく記述の簡便性の向上を実現した. 具体的には, SuperSQL のグルーピングによる文脈を生かす事により, 窓関数を使用する際に SQL で書かなければならない order by や partition by といった窓関数特有の指定の省略を許し, 窓関数になじみの浅い人でも感覚的に正確に用いる事の出来る関数の記法を提案し, 実現した.

キーワード　SuperSQL, SQL, window function

1. 背　　景

近年, ビジネスインテリジェンスにおけるデータの取り扱いの重要性は増大している. データをデータベースから取り出す際に, 窓関数を用いる事により, 複雑なクエリを用いる事なく整理する事が可能となる. 本研究では, 代表的な窓関数を SuperSQL 上に実装し, 通常の SQL の文法と比べて, 汎用性を損なう事なく記述の簡便性の向上を実現した. 具体的には, SuperSQL のグルーピングによる文脈を生かす事により, 窓関数を使用する際に SQL で書かなければならない order by や partition by といった窓関数特有の指定を排除し, 窓関数になじみの浅い人でも感覚的に正確に用いる事の出来る関数の記法を提案し, 実現した.

2. SuperSQL

この章では, 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, 慶應義塾大学の遠山研究室で開発されている [1][4]. 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 [1].

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.

- 垂直結合子 (!)

データを縦に結合して出力.

- 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.

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.

- 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.

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

2.4.1 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(id, path=”./pic”)

2.4.2 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(cou.name, file=”./menu.sql”, att=co.country)

2.4.3 集 約 関 数

TFE における集約関数は、一般の SQL において提供されるものと同じ機能を満たすが、TFE のグルーピング特性を利用し、階層的な集計分類を直観的に記述することが可能である。

```
GENERATE  [店舗名,  ave([給与!]),  sum([売上高!])]!
```

という問い合わせでは、店舗名の後ろに店舗毎の従業員の平均給与、売上合計が、示される。

集約関数には以下のようなものがある。

- max([項目]!)

指定された項目の属性値の最大値を求める。

- min([項目]!)

指定された項目の属性値の最小値を求める。

- sum([項目]!)

指定された項目の属性値の合計を求める。

- ave([項目]!)

指定された項目の属性値の平均を求める。

- count([項目]!)

指定された項目の属性値の個数を求める。

2.4.4 集 約 関 数

TFE における集約関数は、一般の SQL において提供されるものと同じ機能を満たすが、TFE のグルーピング特性を利用し、階層的な集計分類を直観的に記述することが可能である。

```
GENERATE  [店舗名,  ave([給与!]),  sum([売上高!])]!
```

という問い合わせでは、店舗名の後ろに店舗毎の従業員の平均給与、売上合計が、示される。

SuperSQL の集約関数は、SQL の集約関数のように、GROUP BY に頼る事なく、文脈依存でどの範囲を対象とするか決定している。つまり、上記の例であっても、どこに書かれたかによって、ave や sum で対象となる範囲が決定されるので、大括弧の内側のレイアウトで大括弧の外側の範囲を集約する事は不可能である。従来の集約関数では大括弧の内側で外側の範囲を対象とする集約を行うことは通常起きないため、この仕様でまかなえていたが、本研究で提案する窓関数では、使用頻度は極めて少ない事が予想されるが、そのような例外的な処理を行う事が予想される。例えば、学年の中のクラスのレイアウトの中で生徒のテストの点数を表示する場合、その点数が、クラス内順位ではなく、学年順位と共に並べて見たいという様な状況である。本研究では、SuperSQL の文脈を生かし、本来必要である窓関数特有の記述を省く事が可能で、その結果直感的な操作を可能となったが、例外的な需要も満たすように、指定すれば、文脈依存を超えた範囲を対象と出来るような仕様とした部分が、これまでの SuperSQL と比べ、全く新しい部分である。

3. 窓 関 数

3.1 窓関数について

SQL において, 窓関数もしくはウィンドウ関数 (英: window function) は結果セットを 部分的に切り出した領域に集約関数を適用できる, 拡張された SELECT ステートメントである [2]. 集約関数 (GROUP BY) に似ているが, Window 関数では複数の行がまとめられることはなく, 行それぞれが返却される. また, 処理中の行以外の行の値を読み取ることも可能である [3].

Window 関数は以下の構文を使う. 関数 (...) OVER (PARTITION BY ...) : 区間に分割関数 (...) OVER (ORDER BY ...) : 区間ごとに並び替え関数には以下の表に挙げたものと, 通常の集約関数 (count や sum など) が使用できる. lag() や lead() は自己結合 (セルフジョイン; Self-Join) の代わりに使用することで, 処理を大幅に効率化できる場合もある [3].
窓関数は, 1999 年初頭, IBM と Oracle が共同で窓関数を標準 SQL に取り入れる拡張を提案した. そして ANSI の素早い対応によって, これらは SQL − 99 に取り入れられた. IBM はこの仕様の一部を DB2 UDB 6.2 に実装した. これは 1999 年中頃にはある程度商用利用が可能になった. 1999 後半にリリースされた Oracle 8i R2 と DB2 UDB 7.1 は, どちらも窓関数のサポートをより強化した [5][6].

3.2 内 部 処 理

window 関数と集約関数の一番大きな違いは, 集約関数では元集合中のある部分集合と, 結果集合の一つの行が対応する点である. [3] 集約を行う際には GROUP BY によって部分集合を定義する. GROUP BY を省略すると部分集合は元集合全体になる. 一方, window 関数の場合は元集合と結果集合が 1 対 1 で対応するが, 元集合の一行をもとに定義されるある部分集合を使って, 結果集合の一行を定義できる. つまり, 対象行が生成するウィンドウフレームが作り出す部分集合が, 結果となる行を返すので, window 関数では, 元集合と結果集合の行数は変わらない. 元集合と結果集合の行を 1 対 1 で定義したいが, 結果集合の行は元集合の複数の行の値を使って計算したい, という欲求を叶えるのがこの window 関数である [3].

3.3 問 題 点

window 関数を実際に使用する際の問題点は,
クエリの書き方が複雑で, 自在に使いこなせるまでの習得が困難である事である.

この問題を解決するために, 本研究では, 感覚的に理解出来る様仕様を定義した.

4. 提案システム

4.1 実装に関して

window 関数の実装に関しては上記した問題点である, クエ

リの複雑性を一番に考慮し, ユーザーが感覚的に使えるように関数を定義する必要があると考えた. しかしながら, クエリを簡略化しすぎると, その分実行可能な機能の幅が狭くなり, 細かなユーザのニーズを満たせなくなる恐れがある. 例を挙げれば, window 関数は partition by という記述で, 対象となる区間を指定し, その後何について並び替えるのかという指定を order by という記述で行う. この, partition by と order by という記述は, window 関数のクエリの複雑性を一番に高め, ユーザと window 関数がなじみにくくする原因を作っていると考えられる. したがって, 本研究ではこの二つの記述を省略可能にし, ユーザの記述から, どのように区間分けを行い, どのようにソーティングするかを推し量る事につとめた. その上で, 使用頻度が低いと思われる部分に置いては, ユーザがその部分の記述を追記出来るようにオプションとしてもうける形をとった. 詳しくは 4 章で後述する.

表 1 . 窓関数一覧

利用可能なWindow関数

関数	説明
row_number()	行番号
rank()	ランキング (同率で番号を飛ばす)
dense_rank()	ランキング (同率で番号を飛ばさない)
percent_rank()	ランキング (%で表示) : (rank - 1) / (全行数 - 1)
cume_dist()	percent_rank に類似 : (現在の行の位置) / (全行数)
ntile(N)	ランキング (1..N に分割)
lag(value, offset, default)	ソート状態での前の行の値
lead(value, offset, default)	ソート状態での後の行の値
first_value(value)	最初の値
last_value(value)	最後の値
nth_value(value, N)	N番目の値 (1から数える)

図 1 で示した window 関数のうち主立ったいくつかをその使用例と共に紹介する.

4. 2 SuperSQL 窓関数

4. 2. 1 rank 関 数

rank() 関数は, データをある属性でソーティングし, その結果, その値がテーブルの中で何番目の順位に属しているかを表示する関数である.

[l]rank 関数の SQL での使用例 city, name, height, rank() OVER(PARTITION BY city ORDER BY height)

[l]rank() 関数の SuperSQL での使用例 [city, [name, height, rank(height)]!]!

使用例では, 都市に住む人の身長を, 都市ごとに順位付けして表示している. 提案システムでは, partition by という記述を省く代わりに, [ ] の外側に区分けしたい属性を記述する事と定義する. また, order by という記述を省く代わりに, rank() の () の中に書いてある属性で, 自動的にソーティングをして, その結果で rank をふり分ける事とする. つまり, 一般的には

SuperSQL で [value1, [value, rank(value2)]!]!
FROM student_score
と記述する事で,
select value, rank()
over(partition by value1 order by value2)
FROM student_score; という SQL の記述と同じ結果を SuperSQL で返す事とする.

記述方法は dense_rank(), parcent_rank(), cume_dist() と同様である。

4. 2. 2 dense_rank 関数

dense_rank() 関数は, rank をふり分ける際, 同率順位の値があった場合であっても, その順位をつめて表示する関数である.

4. 2. 3 parcent_rank 関数

parcent_rank() 関数は, rank をふり分ける際にその値を百分率で表示する関数である.

4. 2. 4 cume_dist 関数

cume_dist() 関数は, rank をふり分ける際にその値を百分率で表示する関数であるが, parcent_rank() と違い, 最終行の値を 1 にする様に, 現在の行の位置を, 全行数で除する事により求める関数である.

4. 2. 5 ntile 関 数

ntile() 関数は, ある値でソーティングした結果をもとに指定した個数のグループに上位からふり分けるとしたらどのグループに属するかを求める関数である. [l]ntile 関数の SQL での使用例 SELECT name, math, ntile(5) OVER (ORDER BY math DESC) AS ”5 分割” FROM student_score ORDER BY math DESC;

SuperSQL での記述方法は order by の代わりに, ntile() 関数の引数として指定する必要がある.

SuperSQL で
name, value, ntile(5, value, asc)
! もしくは,
name, ntile(5, value1, asc), value1,
ntile(5, value2, asc), value2
!
と記述する事で,
SELECT name, value, ntile(5)
OVER (ORDER BY value)
もしくは
SELECT name, value1, ntile(5)
OVER (ORDER BY value1 ASC)
AS value1 の 5 分割,
value2, ntile(5) OVER(ORDER BY value2 ASC)
AS value2 の 5 分割
FROM student_score ORDER BY value1 ASC;
という SQL の記述と同じ結果を返す事とする.

4. 2. 6 lag 関 数

lag() 関数は, ある値でソーティングした結果, その行のいくつか前の行にある値を持ってくる事が出来る関数である. [l]lag 関数の SQL での使用例 SELECT id , prev_id FROM(SELECT id,
lag(id, 1, 0) over (order by id) as prev_id
From windowharry)as t where id <>

$prev\_id + 1$;

第一引数ではどの値でソートを行うか. 第二引数ではいくつ前の値を持ってくるか. 第三引数では, その値がなかった時になんと表示するかを指定する.

SuperSQL で
[name, value, lag(value, 2, default)
]!
FROM student_score と記述することで,
SELECT name, value, lag(value, 2, default)
OVER(ORDER BY value)
FROM student_score;
と SQL で記述したのと同じ結果を返す事とする.

4.2.7 lead 関 数

lead() 関数は, ある値でソーティングした結果, その行のいくつか後の値にある値を持ってくる事が出来る関数である.

4.2.8 first_value 関数

first_value() 関数は, ある値でソーティングした結果, そのテーブルの最初の行にあたる値を持ってくる事が出来る関数である. [l]first_value 関数の SQL での使用例 select grade, name, math, first_value(math) over(partition by grade order by math) from student_score; partition by の代わりに [ ] を用い, order by を省く代わりに, first_value() の引数でソーティングする. つまり, SuperSQL で
value1, [value2, first_value(value2)
!]! FROM student_score と記述する事で,
SELECT value1, value2, first_value(value2)
OVER(PARTITION BY value1 ORDER BY value2)
FROM student_score;
と SQL で記述したのと同じ結果を返す事とする.

4.2.9 last_value 関数

last_value() 関数は, ある値でソーティングした結果, そのテーブルの最後の行にあたる値を持ってくる事が出来る関数である.

4.2.10 nth_value 関数

nth_value() 関数は, ある値でソーティングした結果, そのテーブルの n 番目の行にあたる値を持ってくる事が出来る関数である.

4.3 提 案 仕 様

以上の SuperSQL の窓関数の仕様をまとめると、以下のようになる。

rank(value [, ”asc” or ”desc”])
dense_rank(value [, ”asc” or ”desc”])
percent_rank(value [, ”asc” or ”desc”])
cume_dist(value [, ”asc” or ”desc”])
ntile(value, 分割数 [, ”asc” or ”desc”])
first_value(value)
last_value(value)
nth_value(value, n [, ”asc” or ”desc”])
lag(value[, num[, default]])
lead(value[, num[, default]])

※ num のデフォルトは 1 、default のデフォルトは 0 であり、asc, desc のデフォルトは asc とした。

over という装飾子を指定することにより、グルーピングの外側の窓関数の結果も取得できるようにした。 ex) [grade, [rank(math), rank(math)@over]!]!

4.3.1 提案システムについての検討

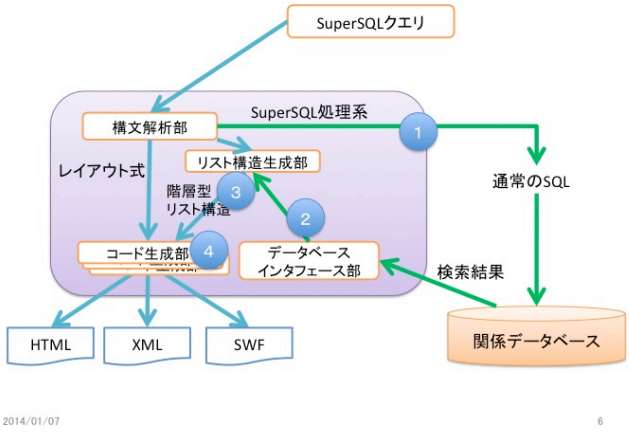

図 1 . 提案システムの内部処理

window 関数を SuperSQL 上で実装するにあたり, どのようにこれらの関数達を設計するかを考えた.

具体的には, 関数の処理をどこで行うかと言う事であり, その選択肢は以下の 4 カ所があり得た.

- parser と SQL 処理の間 (1)
- SQL 処理と tree generator の間 (2)
- tree generator と code generator の間 (3)
- code generator の内部 (4)

(1) で行った場合は, 全ての処理を任せこちら側でする処理は, SuperSQL のどの様な記述が SQL のどの記述に対応するのかを定義していくという作業になる. この場合は, 全てを SQL に依存してしまうため, SQL の仕様によっては, 窓関数が対応していないものも存在するため, 使えない場合も出てくる。(3) で行った場合は, tree を生成後の処理となるので, データそのものを引き出し, 木構造を作り上げたあとに window 関数を適応する事になる. この場合, 窓関数の結果を用いての選択処理が行えないという欠点が存在する. (4) で行った場合も, 同様に, 窓関数の機能を実際にビジネスインテリジェンスとして用いる事を考えた際に選択できる機能が制限されてしまう.

以上の理由から, 本研究では tree generator の前での処理を選択するに至った. しかしながら, 今後窓関数の SQL 側での開発が進み, 一般的にどんな SQL 言語にも対応する日がくれば, 処理速度の点を考慮した場合 (1) が最も望ましいという結論に

至る可能性もあると考えられる．よって，本研究としては (2) の位置を実装として進めるが，(1) の場合どの様な事が行われるかを hand simulation で示していきたい．

5. 評　　価

本システムの有用性を評価するため，SQL の窓関数と SuperSQL の窓関数を両方使って，表を作成してもらう実験を行った．

5.1 実験：アンケート

以前に SQL，SuperSQL の両方を使った事があるユーザ 9 名を対象に，SQL,SuperSQL で 4 種類の窓関数を用い，表を作成してもらい，それぞれの窓関数について，どちらの方が使いやすかったかを「SQL の方がとても使いやすかった,SQL の方がやや使いやすかった,SuperSQL の方がやや使いやすかった,SuperSQL の方がとても使いやすかった」の四択で答えてもらった．また，その理由も答えてもらった．

使用したのは，以下のようなデータベースである．生徒の id，学年，名前，国数英三教科のテストの点数が入っている．また，使用してもらった窓関数は，rank，ntile,lag，frist_value である．

chap3 — psql — 80×45

```
harry=#
harry=#
harry=# select * from student_score ;
 student_id | grade |    name     | math | english | japanese
------------+-------+-------------+------+---------+----------
          1 |     1 | Harry       |   95 |      92 |       90
          2 |     1 | Ron         |   42 |      35 |       41
          3 |     1 | Hermione    |   52 |      95 |       53
          4 |     1 | Fled        |   83 |      11 |       12
          5 |     1 | Geoge       |   46 |      94 |       38
          6 |     1 | Tom         |   55 |      25 |       73
          7 |     1 | Horntail    |   17 |      52 |       72
          8 |     1 | Grim        |   94 |      91 |       94
          9 |     1 | Dippet      |   23 |      22 |       18
         10 |     1 | Kreacher    |   43 |      61 |       48
         11 |     1 | Florver     |   58 |      17 |       96
         12 |     1 | Hermes      |   27 |      47 |       60
         13 |     1 | Pomfly      |   47 |      57 |       60
         14 |     1 | Padma       |   44 |      55 |       99
         15 |     1 | Fang        |    3 |      37 |       22
         16 |     1 | Hagrid      |    6 |      31 |        6
         17 |     1 | Wood        |   46 |      92 |       33
         18 |     1 | Vector      |   78 |      77 |       76
         19 |     1 | Mosag       |   27 |      47 |       44
         20 |     1 | Dnmbledore  |    9 |       6 |        7
         21 |     1 | Molly       |   62 |      21 |       21
         22 |     1 | Slughorn    |   33 |      59 |       28
         23 |     1 | Tomas       |   14 |      97 |       95
         24 |     1 | Crabbe      |   17 |      42 |       29
         25 |     1 | Manticore   |   91 |      83 |       52
         26 |     1 | Shacklebolt |    4 |      26 |       99
         27 |     1 | Dementor    |   26 |      57 |       72
         28 |     1 | Dudley      |   48 |      41 |       63
         29 |     1 | Buckbeak    |   68 |      31 |       83
         30 |     1 | Tonks       |   10 |      63 |       93
         31 |     2 | Rosmerta    |  100 |      46 |       95
         32 |     2 | McLaggen    |   10 |      57 |       92
         33 |     2 | Peeves      |   12 |      57 |       53
         34 |     2 | Binky       |   22 |      72 |       23
         35 |     2 | Patil       |   55 |      17 |        9
         36 |     2 | Werewolf    |   70 |      79 |       89
         37 |     3 | Ptolemy     |   10 |      19 |       29
         38 |     3 | Vernon      |   50 |      49 |       39
         39 |     3 | Dean        |   60 |      69 |       78
         40 |     3 | Dobby       |   84 |      27 |       63
```

実験に使用したデータベース

6. 結果・評価

6.1 rank 関数の使いやすさについて

SuperSQL の rank 関数の有用性について行われたアンケートは次のようなものである．

Q1. rank 関数を用い，英語の点数の高い方から順番にランキングをふる場合，どちらが使いやすかったか．

Q1. のアンケート結果を図 1 に示す．

6.2 ntile 関数の使いやすさについて

SuperSQL の ntile 関数の有用性について行われたアンケートは次のようなものである．

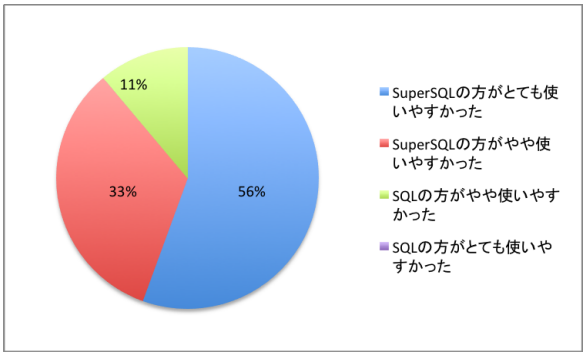

図 1　rank 関数を用い，学年別に英語の点数の高い方から順番にランキングをふる場合，どちらが使いやすかったか．

Q2. ntile 関数を用い，国数英 3 教科の合計をもとに，学年別に成績を 5 段階でつける場合，どちらが使いやすかったか．

Q2. のアンケート結果を図 2 に示す．

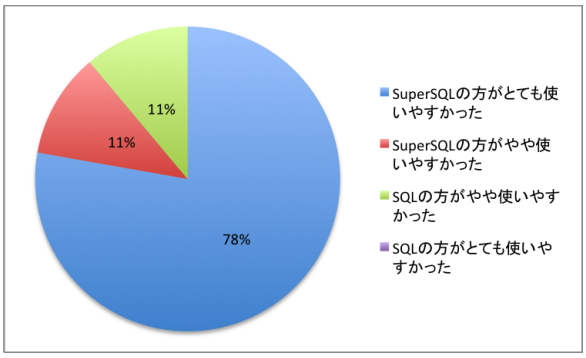

図 2　ntile 関数を用い，国数英 3 教科の合計をもとに，学年別に成績を 5 段階でつける場合，どちらが使いやすかったか．

6.3 first_value 関数の使いやすさについて

SuperSQL の first_value 関数の有用性について行われたアンケートは次のようなものである．

Q3. first_value 関数を用い，数学の点数の一番高い人と自分との点数を並べて表示し，その差を求める場合，どちらが使いやすかったか．

Q3. のアンケート結果を図 3 に示す．

6.4 lag 関数の使いやすさについて

SuperSQL の lag 関数の有用性について行われたアンケートは次のようなものである．

Q4. lag 関数を用い，学年別に国語の点数で高い方から並び替え，自分より順位が一つ上の人と自分の点数を並べて表示し，その差を求める場合，どちらが使いやすかったか．

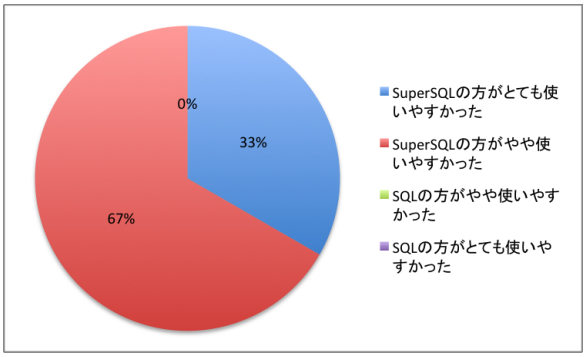

図 3　first_value 関数を用い, 学年別に数学の点数の一番高い人と自分との点数を並べて表示し, その差を求める場合, どちらが使いやすかったか.

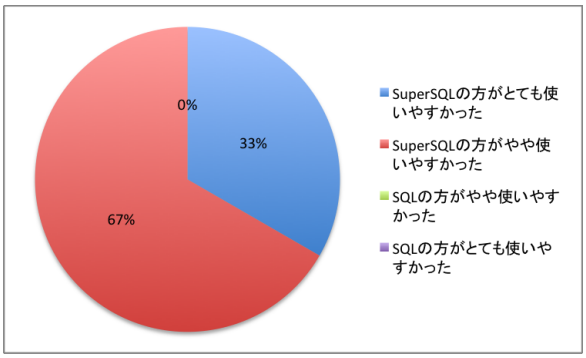

図 4　lag 関数を用い, 学年別に国語の点数で高い方から並び替え, 自分より順位が一つ上の人と自分の点数を並べて表示し, その差を求める場合, どちらが使いやすかったか.

Q4. のアンケート結果を図 4 に示す.

4 つ全ての実験において, SQL に比べて SuperSQL の窓関数が使いやすいと答えた人の割合（SuperSQL の方がとても使いやすかった＋ SuperSQL の方がやや使いやすかった）は 8 割を超えている. この事からも, SQL の窓関数に比べ, SuperSQL の窓関数が使いやすかったと言う事がわかり, 提案システムの有用性が示せたといえると思う. 100%の人が SuperSQL が使いやすいと答えた first$_v$*alue*, *lag* の実験では, 回答理由として「*SQL* の書き方が難しく, とても苦労した.」などの意見が寄せられた.このことからも, *SQL* の窓関数の書き方が *SQL* を使い慣れた人にとってもとても難しいものであることがわかった.

また, 二名のユーザに対して, Q 1 , Q 2 だけではあるが, その作成時間を計った. 結果を, 図 5 に示す.

二つのクエリの作成時間は, どちらも SuperSQL の方が短かった. このことからも, SuperSQL の窓関数の使いやすさがわかる.

7. ま と め

本研究では, 窓関数を SuperSQL 上で実装し, ユーザのニーズに最も適応する仕様を考えてきた. 本研究で, 窓関数の機能を感覚的に SuperSQL 上で用いる事がある程度可能となったと思う. しかしながら, まだその実現範囲には制限があり, 個々の関数にそれぞれどの程度まで詳しくオプションをつけていくかに

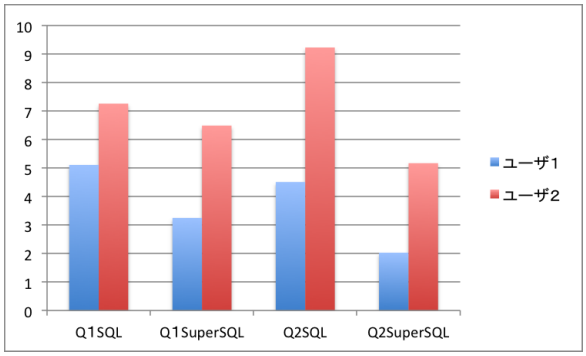

図 5　Q1, Q2 の作成時間の SQL と SuperSQL の比較

関しては, 課題が残ると思う. 今後はそのオプションの充実を考えるとともに, SQL 側での窓関数の充実に合わせ, 処理箇所を変更していくことも考慮していかなくてはならないと思う.

文　献

[1] 石崎文規, 遠山元道. SuperSQL による HTML5 の生成. In *DEIM, 2011*, IEICE
http://db-event.jpn.org/deim2011/proceedings/pdf/d4-1.pdf

[2] Window 関数 mdash;Let's Postgres:
http://lets.postgresql.jp/documents/technical/window_functions

[3] Window 関数 - 導入編 mdash; still deeper:
http://chopl.in/blog/2012/12/01/window-function-tutorial.html

[4] Motomichi Toyama, "SuoerSQL: An Extended SQL for Database Publishing and Presentation, " Proceedings of ACM SIGMOD '98 International Conference on Management of Data, pp. 584-586, 1998

[5] SQL ゼロからはじめるデータベース操作
著者　ミック
発行所　株式会社翔泳社 (http://www.shoeisha.co.jp)

[6] プログラマのための SQL 第 4 版 すべてを知り尽くしたいあなたに
著者 ジョー セルコ
監訳 ミック
発行所　株式会社翔泳社 (http://www.shoeisha.co.jp)