集計レポート

牧之内研 M1 龍 浩志

2001 年 6 月 22 日

1 集計レポート

データベースでは，問い合わせを行なって目的のデータを検索するというだけでなく，データベースに登録されているデータや検索結果をいろいろな角度から分析をすることができるようにデータの集計を行なうことができます．例えば, 売り上げテーブルから売り上げの合計を集計する，得意先ごとに売り上げを集計するなどです．そこで，今回はこのような集計結果をレポートするために必要となる集約関数や問い合わせについて，また Jasmine での例について説明します.

2 集約関数

複数のタプルを対象として，タプルの数，合計，平均，最大値，最小値といった集計値を求める関数を集約関数といいます． SQL では,

- count()
- sum(属性名)
- avg(属性名)
- max(属性名)
- min(属性名)

などがあり， count() はタプルの数を，それ以外は指定した属性について集約値を求めます．さらに， where 句の条件を使って絞り込むと，絞り込んだタプルのみを対象として，集約値を求めることができます.

<売り上げテーブル>

商品番号	商品名	売り上げ
1	Apple	50
2	Orange	100
3	Grape	150

タプルの数

```
select count()
from 売り上げテーブル
```

→ 3

売り上げ総額

```
select sum(売り上げ)
from 売り上げテーブル
```

→ 300

Jasmine の場合は， count(),sum(),average(),min(),max() という関数が用意されています． count() は任意の集合 p1 に対して,

```
p1.count();
```

として，集合 p1 の要素数を求めることができますが, その他の関数は数値型の集合 (Integer,Decimal,Real) に対してのみ行なえます．集合 p2={50,100,150} のような数値型の集合に対して,

```
p2.sum();
p2.average();
p2.min();
p2.max();
```

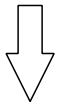
として，各種の集約値を求めることができます.

3 group-by

group-by 句は特定の列に同じ値をもつものをまとめて，グループ化を行ないます．さらに，集約関数を組み合わせて各グループごとの集約値を計算することができます．下の図は，売り上げテーブルから商品ごとにグループ化を行ない，商品ごとの売り上げを求めています．結果は，グループ化した属性と集約値の２つの列となります.

売上テーブル(Salesクラス)

salesNo (売上番号)	soldCommodityNo (商品番号)	purchaseNo (得意先番号)	quantity (量)	amountSold (売上総額)
1	2	1	10	200
2	1	2	20	600
3	3	3	10	400
4	2	2	30	600
5	1	3	10	300



select　商品番号 、sum(売上総額)
from　売上テーブル
group-by 商品番号

soldCommodityNo (商品番号)	amountSold (売上総額)
2	800
1	900
3	400

Jasmine で記述する場合,

```
> Bag<[Integer CommodityNo,Integer TotalSales]> g;
```

（グループ化する属性，集約した結果）という形のタプルの集合変数 g を用意します.

```
> g = group s in Sales by (s.soldCommodtiyNo)
     with (partition^amountSold.sum());
```

- `group s in Sales`

  Sales クラスのオブジェクト s について group 化する． s は範囲変数といいます.

- `by (s.soldCommodityNo)`

  by の後に group 化を行なう属性名を指定します．オブジェクト s の soldCommodityNo によって group 化する.

- `with (partition^amountSold.sum())`

  with の後には，各 group に対し，どのような集約計算を行なうかを指定します． group 化された後の個々の集合を partition として扱います． partition に対し， amountSold の合計を求めます． partition の要素数を求める場合は， partition.count() とします.

各 partition に対し集約計算を行なわず，実際の要素を集合として出す場合は，

```
partition^amountSold
```

とします．取り出した結果は下の表のようになります．この場合最初に宣言するタプルの中で集合であることを指定しておく必要があります

```
Bag<[Integer CommodityNo,Bag<Integer> TotalSales]> g;
```

soldCommodityNo (商品番号)	amountSold (売上総額)
2	200 , 600
1	600, 300
3	400

4 order-by

order-by 句を使うとデータベースからデータを取り出すときに，順序を並べ替えて取り出すことができます． order-by 句で指定した属性に従って，その属性値の順で，昇順または降順で並べ替えることができます． SQL では，

select 取り出したい属性名
from テーブル
order by ソートを行なう属性名

という記述ができます． Jasmine で並べ替えを行なうには， sort() 関数を使用します． sort() は，単一型の集合，複合型の集合に使えます．昇順 (ASCEND), 降順 (DESCEND) を指定することができ, 指定しなかった場合は昇順になります.

- 単一型の集合の場合

```
> Bag<Integer> i;
> i={4,7,6,3};
> i.sort().print();

  Bag{ 3, 4, 6, 7 }

> i.sort(DESCEND).print();

  Bag{ 7, 6, 4, 3 }
```

- 複合型の集合の場合
  この場合， sort() の引数として，ソートを行なう属性名を指定する必要があります.

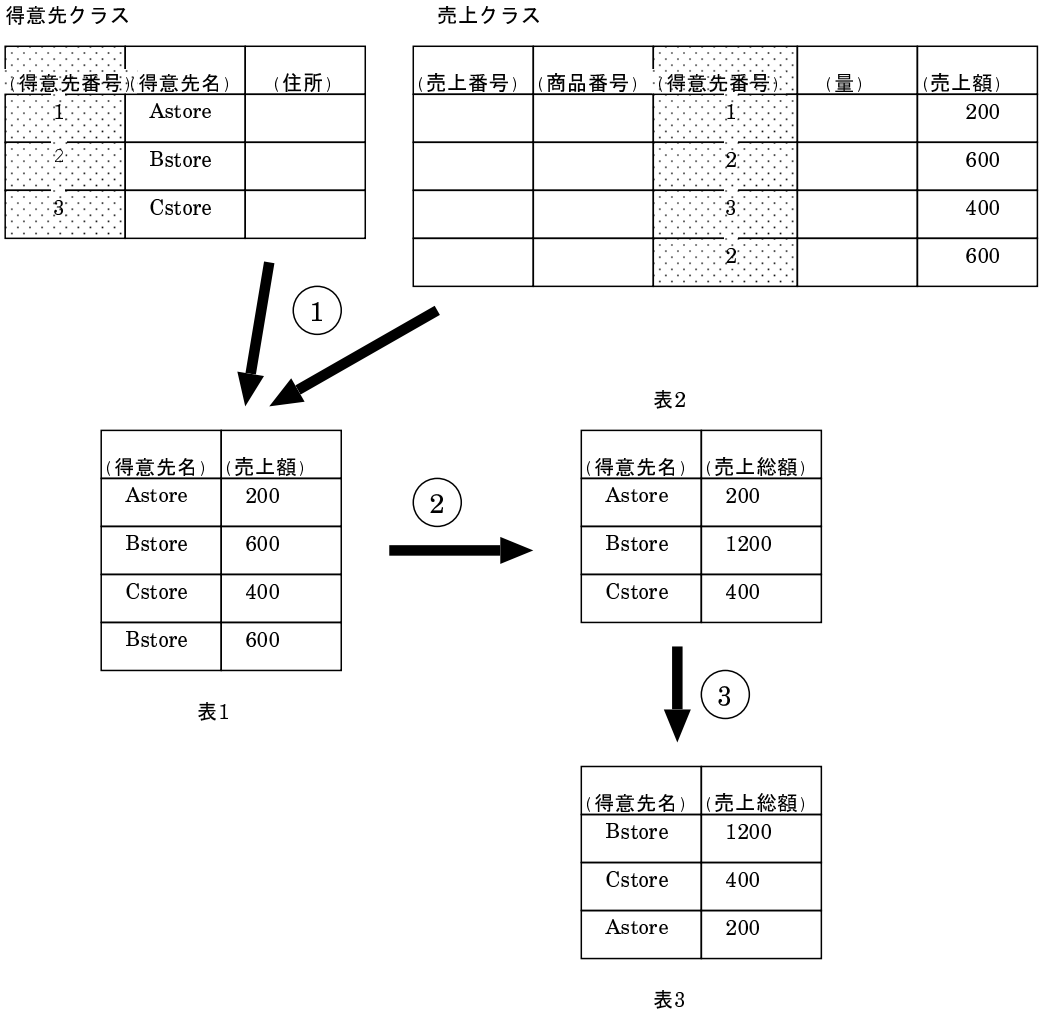
```
>Bag<Sales> s;
>s = Sales from Sales;
>s.sort(‘‘amountSold’’).print();

>s.sort(‘‘amountSold’’,DESCEND).print();
```

注：上の例のようにある集合 A に対し， A.sort() としても，元の集合 A は実際には並べ替えられません． A.sort().print() というのは，集合 A をソートした順に表示するという意味です．並べ替えを元の集合に反映させるには， A=A.sort() とし，元の集合に代入してやる必要があります.

5　問い合わせ例

これまで，見てきた問い合わせ (selection,projection,join,group-by,order-by) を組み合わせることによりより複雑な集約計算を行なうことができます．以下では次のようなクエリについて考えてみます．“得意先ごとの売り上げを売り上げが多い順に，その得意先名と売り上げを出力する”



1. Join , projection

   得意先クラス，売上クラスを得意先番号によって Join を行ない，新しくできた表から得意先名，売上額を projection によって取り出します．(表 1)

   ```
   > Bag<[String customer,Integer sales]> result;

   > result = [Customer.customerName,Sales.amountSold] from Customer, Sales
     where Customer.customerNo == Sales.purchaserNo
   ```

2. group-by
   1 の結果の表に対し，得意先名によってグループ化し，各グループごとの売り上げの合計 (sum) を求めます．(表 2)

   ```
   > result = group g in result by (g.customer) with (partition^sales.sum());
   ```

3. order-by
   2 の結果を売上総額の多い順 (降順) に並べ替えます．(表 3)

```
> result = result.sort(‘‘sales’’,DESCEND);
```

注：問い合わせの結果を求める手順は１つとは限りません．例えば，上では１つの例を示しましたが，最初に group-by をした後に Join を行なうこともできます．

6 実習

group-by,order-by，さらに selection,projection,join を組み合わせて問い合わせを行ないましょう． Commodity クラス， Customer クラス， Sales クラスにサンプルデータを挿入するファイルを /u/ryu/work/rinkou/database/querydata.odql に置いときます．自分でデータを作って挿入しても構いません．以下のような問い合わせをやってみてください．

- 問い合わせ例

  1. 得意先ごとに何を売ったか
     結果例

     ```
     Bag{
       [store: "Astore", fruits: Bag{ "Orange", "Grape", "Mellon" }],
       [store: "Bstore", fruits: Bag{ "Orange", "Banana", "Apple" }],
       [store: "Csotre", fruits: Bag{ "Grape", "Mellon", "Apple", "Orange" }]
     }
     ```

  2. 得意先ごとに売り上げの平均値を求め，多い順に出力
     結果例

     ```
     Bag{
       [store: "Astore", salesAverage: 1066],
       [store: "Csotre", salesAverage: 600],
       [store: "Bstore", salesAverage: 466]
     }
     ```