情報科学演習 資料 11
演算子・関数とテーブルのグループ化

2020 年 7 月 25 日

目 次

1 演算子と関数

SELECT 文で利用可能な演算子・関数の幾つかとそれらの使い方を紹介する。これらはすべて SELECT に続く列名指定の箇所，および WHERE 句で利用できる。なお，データベースシステムへの問い合わせの基礎 — SQL と SELECT 文の基本の第 3.3 節で紹介済みの比較演算子や論理演算子も同様である。

1.1 算術演算子

数値データに対して演算を施すために，次の算術演算子が使える。

- `+` 加算
- `-` 減算
- `*` 乗算
- `/` 除算
- `%` 剰余

これらの優先順位は数式での演算順位と同じであり，() を使って演算の順序を変更できる。

次の例はテーブル population の全ての内容に加え，列 popul を 1000 で割った値を出力する。

```
SELECT *, popul/1000 FROM population;
  name   | popul  | house  | ?column?
---------+--------+--------+----------
 函館市  | 265979 | 123950 |      265
 北斗市  |  46390 |  18508 |       46
 七飯町  |  28120 |  11141 |       28
 鹿部町  |   4226 |   1660 |        4
 森町    |  15946 |   6628 |       15
```

さらに，列を限定するとともに，算術演算子を含む式を用いて行を限定する。

```
SELECT name, popul/1000 AS sennin FROM population
WHERE popul/1000 = 28;
  name   | sennin
---------+--------
 七飯町  |     28
```

1.2 文字列演算子

文字列演算子の一つとして，文字列を連結する演算子 || を紹介する。次の SQL 文は，列の内容に (hakodate) を結合して出力する。

```
SELECT name || '(hakodate)' FROM spring WHERE area = '01202';
        ?column?
------------------------
 谷地頭温泉(hakodate)
```

湯の川温泉街 (hakodate)
川汲温泉郷 (hakodate)
戸井温泉 (hakodate)

次の SQL 文では二つの列の内容を連結する。

```
SELECT new_post_code AS postcode, town_kanji || zone_kanji AS shi_cho
FROM postcode
WHERE new_post_code = '0400083';
 postcode |   shi_cho
----------+--------------
 0400083  | 函館市八幡町
```

1.3 関数

文字列の長さを求める関数 CHAR_LENGTH を紹介する。この関数の戻り値は整数 (integer 型) である。

以下は，name 列の内容の文字数を出力する例と，文字数で行を限定する例である。

```
SELECT *, CHAR_LENGTH(name) FROM spring ;
     name      | area  | char_length
---------------+-------+-------------
 谷地頭温泉    | 01202 |           5
 湯の川温泉街 | 01202 |           6
 東大沼温泉郷 | 01337 |           6
 川汲温泉郷    | 01202 |           5
 戸井温泉      | 01202 |           4
 せせらぎ温泉 | 01236 |           6
 鹿部温泉郷    | 01343 |           5
 濁川温泉郷    | 01345 |           5
 仁山温泉      | 01337 |           4

SELECT * FROM spring WHERE CHAR_LENGTH(name) = 5;
    name     | area
-------------+-------
 谷地頭温泉 | 01202
 川汲温泉郷 | 01202
 鹿部温泉郷 | 01343
 濁川温泉郷 | 01345
```

なお，CHAR_LENGTH の引数には，CHAR_LENGTH(' 函館') のように定数を与えることも可能である。

1.4 参考 — FROM 句の省略

テーブル内のデータを必要としない演算結果等を求めるときには FROM 句を省略できる。

```
select 1 + 2;
select '函館' || '市';
select CHAR_LENGTH('函館');
```

2 集合関数とテーブルのグループ化

2.1 集合関数

集合関数 (set function) は，データに関する集計計算を行うためのものである。集合関数を集約関数 (aggregate function) や集計関数と呼ぶことがある。

```
COUNT(*)  行数 (*の代わりに列名指定も可)
SUM(列名) 合計
AVG(列名) 平均
MAX(列名) 最大値
MIN(列名) 最小値
```

population テーブルの内容は次のとおりである。

```
SELECT * FROM population ;
  name  | popul  | house
--------+--------+--------
 函館市 | 265979 | 123950
 北斗市 |  46390 |  18508
 七飯町 |  28120 |  11141
 鹿部町 |   4226 |   1660
 森町   |  15946 |   6628
```

このテーブルに対して，次のように集合関数を適用できる。

```
SELECT COUNT(*), MAX(popul), SUM(popul), MAX(house), SUM(house)
FROM population WHERE popul < 100000;
 count |  max  |  sum  |  max  |  sum
-------+-------+-------+-------+-------
     4 | 46390 | 94682 | 18508 | 37937
```

集合関数は WHERE 句の検索条件には使えない。また，集合関数の値とテーブル内の値を同時に表示するときには，次節に示す GROUP BY を使ったテーブルのグループ化が必要である。

```
SELECT name, COUNT(*) FROM population;
ERROR:  column "population.name" must appear in the GROUP BY clause
or be used in an aggregate function
ERROR:  列"population.name"は GROUP BY 句で出現しなければならないか、集約
関数内で使用しなければなりません。
```

2.2 テーブルのグループ化

2.2.1 GROUP BY

GROUP BY は，同じ値を持つ行を元にテーブルをグループ化して扱うために使う。グループ化された各行に対しては，集合関数を適用することができる。

まず，これまでに紹介した方法で，spring テーブルの内容全てを area 列で昇順に並べ替えて表示する。

```
SELECT * FROM spring ORDER BY area;
     name      | area
---------------+-------
 谷地頭温泉    | 01202
 湯の川温泉街  | 01202
 川汲温泉郷    | 01202
 戸井温泉      | 01202
 せせらぎ温泉  | 01236
 東大沼温泉郷  | 01337
 仁山温泉      | 01337
 鹿部温泉郷    | 01343
 濁川温泉郷    | 01345
```

これを area 列でグループ化して，各グループに同じエリアの行が何行ずつあるのかを表示する SQL と実行結果は，次の通りとなる。表示順は，行数の多い順（降順）としている。

```
SELECT area, COUNT(*) FROM spring
GROUP BY area
ORDER BY count DESC;
 area  | count
-------+-------
 01202 |     4
 01337 |     2
 01345 |     1
 01343 |     1
 01236 |     1
```

一般に，GROUP BY 句を含む SELECT 文において，SELECT の後に指定できる列名は，各グループに対して値が一意に定まる列に限られる。

ただし，この授業で使用している PostgreSQL 8.4 の場合，SELECT の後には GROUP BY で用いた列のみ指定可能である。

```
SELECT name, area FROM spring GROUP BY area;
ERROR:  column "spring.name" must appear in the GROUP BY clause
or be used in an aggregate function
```

2.2.2 HAVING

GROUP BY 句でグループ化したテーブルに対して検索条件を指定するには HAVING 句を使う。集合関数は WHERE 句には使えないが，HAVING 句には使える。

```
SELECT area, COUNT(*) FROM spring
GROUP BY area HAVING COUNT(*) > 1;
 area  | count
-------+-------
 01337 |     2
 01202 |     4

SELECT area, COUNT(*) FROM spring
GROUP BY area HAVING area LIKE '012%';
 area  | count
-------+-------
 01202 |     4
 01236 |     1
```

GROUP BY 句や HAVING 句は WHERE 句と共に使うこともできる。WHERE 句での検索条件が GROUP BY でグループ化される以前の各行を特定するのに対し，HAVING 句での検索条件はグループ化された後の行を特定することに注意せよ。

```
SELECT * FROM area_code, spring
WHERE area_code.code = spring.area;
  name  | code  |     name     | area
--------+-------+--------------+-------
 函館市 | 01202 | 谷地頭温泉   | 01202
 函館市 | 01202 | 湯の川温泉街 | 01202
 函館市 | 01202 | 川汲温泉郷   | 01202
 函館市 | 01202 | 戸井温泉     | 01202
 北斗市 | 01236 | せせらぎ温泉 | 01236
 七飯町 | 01337 | 東大沼温泉郷 | 01337
 七飯町 | 01337 | 仁山温泉     | 01337
 鹿部町 | 01343 | 鹿部温泉郷   | 01343
 森町   | 01345 | 濁川温泉郷   | 01345

SELECT area_code.name AS area_name, COUNT(*)
FROM area_code, spring WHERE area_code.code = spring.area
GROUP BY area_code.name;
 area_name | count
-----------+-------
 森町      |     1
 七飯町    |     2
 鹿部町    |     1
```

```
 函館市     |     4
 北斗市     |     1
```

```
SELECT area_code.name AS area_name, count(*)
FROM area_code, spring
WHERE area_code.code = spring.area
GROUP BY area_code.name HAVING count(*) > 1;
 area_name | count
-----------+-------
 七飯町     |     2
 函館市     |     4
```

3 SELECT 文の書式

SELECT に続いて記述するキーワードや句の順序は定められており，それに違反すると syntax error が起きる。SELECT 文の記述の書式は，psql のヘルプコマンド `\h` を使って調べることができる。

```
\h select
Command:     SELECT
Description: retrieve rows from a table or view
Syntax:
SELECT [ ALL | DISTINCT [ ON ( expression [, ...] ) ] ]
    * | expression [ AS output_name ] [, ...]
    [ FROM from_item [, ...] ]
    [ WHERE condition ]
    [ GROUP BY expression [, ...] ]
    [ HAVING condition [, ...] ]
    [ { UNION | INTERSECT | EXCEPT } [ ALL ] select ]
    [ ORDER BY expression [ ASC | DESC | USING operator ] [, ...] ]
    --- 以下省略 ---
```

ここで `[ ]` 内の要素は省略可能であることを意味し，`|` で区切られた要素はそのうちの何れかを指定できることを意味する。

4 問題

1. テーブル population 内の各自治体について，1 世帯当りの平均人数を求めなさい。関数 CAST を用いて int 型の列を float 型等に一時的に型変換（キャスト）すれば，小数点以下を含めて平均人数を求められる。例えば，popul 列を float 型にキャストするには，列名 popul に代え `CAST(popul AS float)` を使う。出力は次のようにすること。ただし，行の出力順は問わない（以下も指示がない場合は同様）。

```
  name   | popul  | house  |     ?column?
---------+--------+--------+------------------
 函館市  | 265979 | 123950 | 2.14585720048407
 北斗市  |  46390 |  18508 | 2.50648368273179
 七飯町  |  28120 |  11141 | 2.52401041199174
 鹿部町  |   4226 |   1660 | 2.54578313253012
 森町    |  15946 |   6628 | 2.40585395292698
(5 行)
```

2. 平成 12 年度国勢調査数によれば，函館市の人口は 287637 人，世帯数は 121779 世帯である。テーブル population (平成 27 年度時の人口，世帯数）を使って，平成 27 年度の函館市の人口と世帯数が，平成 12 年度からどれだけ増えたかを求めなさい。その際，函館市の人口が 250000 より大きいことを使って，函館市についてのみ次のとおりに出力しなさい。

```
  name   | population | house
---------+------------+-------
 函館市  |     -21658 |  2171
```

3. テーブル area_code と spring から，次の結果を得なさい。行の順序は異なってよい。テーブルの結合と文字列の連結演算子，および列の別名を使う。

```
      loc_name
--------------------
 函館市谷地頭温泉
 函館市湯の川温泉街
 函館市川汲温泉郷
 函館市戸井温泉
 北斗市せせらぎ温泉
 七飯町仁山温泉
 七飯町東大沼温泉郷
 鹿部町鹿部温泉郷
 森町濁川温泉郷
(9 行)
```

4. テーブル postcode において，旧郵便番号 (列名：old_post_code) が 041 である行が何行あるかを出力しなさい。

5. 次の SELECT 文の意味を考え，実行結果を確認しなさい。

```
SELECT town_kanji, old_post_code, new_post_code FROM postcode
WHERE old_post_code LIKE '041%' order by new_post_code;
```

続いて，旧郵便番号が 041 で始まる各行について，同じ市町村名の行が何行ずつあるのか求めなさい。次の結果を得ること。なお，行の順序は異なってよい。

```
  town_kanji  | count
--------------+-------
```

```
函館市          |    79
亀田郡七飯町    |    22
茅部郡鹿部町    |     6
北斗市          |    18
```

6. 前の問題の結果のうち，count の値が 20 以上の行のみを出力しなさい。ただし，各行は count の値が大きい順に出力すること。