

情報科学演習 資料 11

行の並び順, データ型, テーブルの結合

令和5年6月26日

目次

| | | |
|----------|--|-----------|
| 1 | SELECT 文における行の並び順指定: ORDER BY | 1 |
| 1.1 | 昇順指定 | 1 |
| 1.2 | 逆順指定 | 1 |
| 1.3 | 複数列による並び順指定 | 1 |
| 1.4 | WHERE 句を含む SELECT 文での ORDER BY | 2 |
| 2 | データ型 | 2 |
| 2.1 | データ型とは | 2 |
| 2.2 | SQL 文における値の記述法 | 3 |
| 2.3 | データ型は列毎に決められている | 3 |
| 2.4 | 数字だから数値とは限らない | 4 |
| 3 | テーブルの結合 | 5 |
| 3.1 | 準備 | 5 |
| 3.2 | 交差結合 | 6 |
| 3.3 | 等結合 | 7 |
| 3.3.1 | 等結合 | 7 |
| 3.3.2 | 参考 — JOIN ON | 7 |
| 3.3.3 | 参考 — 自然結合 (NATURAL JOIN) | 8 |
| 3.3.4 | 列と行の限定 | 8 |
| 4 | 別名の利用 | 9 |
| 4.1 | 列の別名 | 9 |
| 4.2 | 相関名 (テーブルの別名) | 9 |
| 5 | 問題 | 10 |

1 SELECT 文における行の並び順指定: ORDER BY

これまでの形式で SELECT 文を実行した場合、出力される行の順序は保証されません。行の順序を指定するには ORDER BY を使います¹。

1.1 昇順指定

ORDER BY 句の最も単純な形式は次のとおりです。

```
ORDER BY 列名
```

この形式では、出力行が指定した列名で昇順に並びます。昇順 (ascend) 並びであることを明示して

```
ORDER BY 列名 ASC
```

とすることもできます。

```
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 |

(9 rows)

1.2 逆順指定

行を降順 (descend) に出力するには

```
ORDER BY 列名 DESC
```

とします。

1.3 複数列による並び順指定

列名を、(カンマ) 区切りで複数指定して

```
ORDER BY 列名 1, 列名 2, ...
```

¹関係データベースの行は、理論上は数学における集合の要素に対応します。集合では要素の並び順に意味はありませんので、理論上は関係データベースの行の並び順も無意味です。ただし、実用上はそれでは困りますので、ORDER BY で行の表示順序を指定できるようになっています。

とすれば、列名 1 の値が等しい行が、さらに列名 2 で並べ替えられます。各列名に続き ASC や DESC も指定できます。

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

ここで、name 列の並べ替えは、日本語の文字コードに基づいて行われます。

1.4 WHERE 句を含む SELECT 文での ORDER BY

SELECT 文に「WHERE 検索条件」を含む場合、ORDER BY はその後ろに書く必要があります。

```
SELECT * FROM area_code
WHERE code = '01236' OR code = '01202' OR code = '01337'
ORDER BY code;
  name      | code
-----+-----
函館市    | 01202
北斗市    | 01236
七飯町    | 01337
(3 rows)
```

2 データ型

2.1 データ型とは

テーブル population には、函館市周辺の自治体の名前、人口、世帯数が入っています (平成 27 年国勢調査時のもの)。

```
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
```

このテーブルでは、列 name (自治体名) が文字列であるのに対し、列 popul (人口) と列 house (世帯数) は整数です。文字列や整数のような値の種類を**データ型**といいます。なお、計算機科学では、「値」という言葉は数値以外のもの (文字など) に対しても使われます。

この資料では、SQL に用意されているデータ型の詳細には触れませんが、データ型には文字のための型と数値のための型があることは覚えておいてください。

2.2 SQL 文における値の記述法

SQL ではデータ型に応じて値の指定方法が異なります。

1. 文字を指定するには、それを ' (単一引用符) で囲む。

これまでに使ってきた方法です。このルールに反すると、次のようなエラーが起きます。

```
SELECT * FROM population WHERE name = 函館市;

ERROR: 列"函館市"は存在しません
行 1: select * from population where name = 函館市;
                                         ^
```

(函館市を ' で囲まなかったので列名とみなされた)

2. 数値等の文字以外の値は ' で囲まない。

```
SELECT * FROM population WHERE popul > 200000;
  name | popul | house
-----+-----+-----
  函館市 | 265979 | 123950
```

2.3 データ型は列毎に決められている

テーブルの各列に格納されるデータのデータ型は、テーブルの設計・作成の段階で、列毎に定められています。

psql には、テーブルの情報を表示する、psql 固有のコマンド

```
\d テーブル名
```

があります。これを使うとテーブル population における各列のデータ型がわかります。\\d は SQL 文ではないので、**末尾に ; を付けない**ことに注意してください。

```
db_a=> \d population
```

```

          テーブル "public.population"
  列      |      型      | 照合順序 | Null 値を許容 | デフォルト
-----+-----+-----+-----+-----
  name    | character varying |          |                |
  popul   | integer           |          |                |
  house   | integer           |          |                |

```

これより、テーブル population では、列 name は character varying (可変長文字列) 型であり、列 popul と列 house は integer (整数) 型であることがわかります。したがって、テーブル population では、列 name のデータを SQL で指定する場合には' (単一引用符) で囲む必要があり、列 popul と house ではその必要はありません。

2.4 数字だから数値とは限らない

テーブル area_code の中身は次のとおりです。

```
SELECT * FROM area_code;
```

```

  name | code
-----+-----
  函館市 | 01202
  北斗市 | 01236
  七飯町 | 01337
  鹿部町 | 01343
  森町   | 01345

```

一見、列 name は文字の型に、列 code は数値の型に見えますが、実は違います。

```
db_a=> \d area_code
```

```

          テーブル "public.area_code"
  列      |      型      | 照合順序 | Null 値を許容 | デフォルト
-----+-----+-----+-----+-----
  name    | character varying |          |                |
  code    | character(5)       |          |                |

```

列 name は character varying (可変長文字列) 型で、列 code は character(5) (5文字からなる固定長文字列) 型です。つまり、数字のみが格納された列であっても、その列が数値の型とは限りません。

SQL 文における単一引用符の必要性は、列のデータ型によって決まりますので、テーブル area_code における列 code の値を指定する場合には、' (単一引用符) で囲む必要があります。

```
SELECT * FROM area_code WHERE code = '01202';
```

```

  name | code
-----+-----
  函館市 | 01202
(1 row)

```

```
SELECT * FROM area_code WHERE code = 01202;
```

```
ERROR: 演算子が存在しません: character = integer
行 1: SELECT * FROM area_code WHERE code = 01202;
```

```
HINT: 指定名称、指定引数型に合う演算子がありません。明示的な型キャストが必要かもしれませ
```

なお、列 code (地域コード) を整数型にしない理由として、次のことが挙げられます。

- 特定の桁の数字で検索条件を指定する場合、文字列として扱った方が便利である。like を使った文字列パターンの検索は、文字列に対する操作なので、code を整数型にすると like による検索は (原則として) できない。
- 数値ならば、数値演算 (例えば加算等) の結果は、意味のある数値として解釈できるべきである。しかし、地域コードを整数としても演算の結果には数値としての意味はなく、地域コードを整数にする必要性はない。

ここで扱う地域コードはすべて 5 桁に決まっているので、列 code は文字 5 つから成る固定長文字列としています。

3 テーブルの結合

3.1 準備

この章では、複数のテーブルから意味のある結果を得るための方法を学びます。説明に使うテーブルは次の二つです。

1. テーブル area_code

列 name (地域名): 可変長文字型 (varchar),
列 code (地域コード): 固定長文字型 (char(5))

```
SELECT * FROM area_code ;
```

```
name | code
-----+-----
函館市 | 01202
北斗市 | 01236
七飯町 | 01337
鹿部町 | 01343
森町 | 01345
```

(5 行)

2. テーブル n.hospital

列 code (地域コード): 固定長文字型 (char(5)),
列 num (地域の病院数²): 整数型 (int)

²一般診療所数と病院数の合計

```

SELECT * FROM n_hospital ;
  code | num
-----+-----
  01202 | 204
  01236 |  23
  01337 |  18
  01343 |   2
  01345 |   9
(5 行)

```

この二つのテーブルは、地域コードの入った列によって互いに関連づけられています。例えば、「函館市の病院数が 204 である」ことは、二つのテーブルを利用しなければわかりません。本章の目的は、SELECT 文を使って、このような結果を得る方法を習得することです。

3.2 交差結合

SELECT 文において、FROM に続くテーブルの名前を複数指定することができます。

```
FROM テーブル 1, テーブル 2, ...
```

これにより、FROM に続いて指定したテーブル (テーブル 1, テーブル 2, ...) に含まれる行の、すべての組み合わせが得られます。これを交差結合 (直積³) と呼ぶことがあります。

```

SELECT *
FROM area_code, n_hospital;

  name | code | code | num
-----+-----+-----+-----
  函館市 | 01202 | 01202 | 204
  北斗市 | 01236 | 01202 | 204
  七飯町 | 01337 | 01202 | 204
  鹿部町 | 01343 | 01202 | 204
  森町   | 01345 | 01202 | 204
  函館市 | 01202 | 01236 |  23
  北斗市 | 01236 | 01236 |  23
  七飯町 | 01337 | 01236 |  23
  鹿部町 | 01343 | 01236 |  23
  森町   | 01345 | 01236 |  23
  函館市 | 01202 | 01337 |  18
  北斗市 | 01236 | 01337 |  18
  七飯町 | 01337 | 01337 |  18
  鹿部町 | 01343 | 01337 |  18
  森町   | 01345 | 01337 |  18

```

³出力の各行は集合論における順序対 (または順序 n 組) に対応します。それを全て集めたものは直積です。

```

函館市 | 01202 | 01343 | 2
北斗市 | 01236 | 01343 | 2
七飯町 | 01337 | 01343 | 2
鹿部町 | 01343 | 01343 | 2
森町   | 01345 | 01343 | 2
函館市 | 01202 | 01345 | 9
北斗市 | 01236 | 01345 | 9
七飯町 | 01337 | 01345 | 9
鹿部町 | 01343 | 01345 | 9
森町   | 01345 | 01345 | 9
(25 行)

```

3.3 等結合

3.3.1 等結合

先の交差結合の結果を元に地域毎の病院数を得るためには、WHERE による検索条件指定を行って、二つのテーブル area_code と n_hospital に含まれる列 code の値が同じ行だけを抽出します。複数のテーブルから列を指定する場合、

テーブル名.列名

の形で、テーブルと列を . (ドット) で区切って記述します。

```

SELECT *
FROM area_code, n_hospital
WHERE area_code.code = n_hospital.code;

```

```

  name | code | code | num
-----+-----+-----+-----
函館市 | 01202 | 01202 | 204
北斗市 | 01236 | 01236 | 23
七飯町 | 01337 | 01337 | 18
鹿部町 | 01343 | 01343 | 2
森町   | 01345 | 01345 | 9
(5 行)

```

このように、一方の列の値と、もう一方の列の値が等しい行をつなぎ合わせる操作を等結合といいます。

3.3.2 参考 — JOIN ON

この操作は JOIN と ON という SQL キーワードを使って、次のように書いて実現することもできます。


```
SELECT *
FROM area_code JOIN n_hospital
ON area_code.code = n_hospital.code;
```

3.3.3 参考 — 自然結合 (NATURAL JOIN)

同名の列で等結合した結果から、重複する列を省いた結合を自然結合といいます。area_code と n_hospital の自然結合は

```
SELECT *
FROM area_code NATURAL JOIN n_hospital;
```

で得られます。

3.3.4 列と行の限定

等結合の結果から、特定の列を指定した順序で得たい場合には、これまでと同様に SELECT の後に列名を明示します。

```
SELECT area_code.code, area_code.name, n_hospital.num
FROM area_code, n_hospital
WHERE area_code.code = n_hospital.code;
```

```
code | name | num
-----+-----+-----
01202 | 函館市 | 204
01236 | 北斗市 | 23
01337 | 七飯町 | 18
01343 | 鹿部町 | 2
01345 | 森町 | 9
(5 行)
```

さらに検索条件を加えることにより、取り出す行を限定できます。

```
SELECT area_code.code, area_code.name, n_hospital.num
FROM area_code, n_hospital
WHERE area_code.code = n_hospital.code
AND area_code.code = '01202';
```

```
code | name | num
-----+-----+-----
01202 | 函館市 | 204
(1 行)
```

さらに表示する列と行を限定してみます。

```
SELECT area_code.name, n_hospital.num
FROM area_code, n_hospital
WHERE area_code.code = n_hospital.code
      AND (area_code.code = '01202' OR area_code.code = '01337');
```

```
name | num
-----+-----
函館市 | 204
七飯町 | 18
(2 行)
```

この SQL 文においては、次の点に注意してください。

1. 表示しない列 (code) も検索条件に含めることが可能
2. AND は OR より評価の優先順位が高いため、必要に応じて括弧を使う

4 別名の利用

4.1 列の別名

テーブルを結合すると、表示した列名の意味がわからなくなることがあります。例えば、先の例では、name や num が何の名前や数なのか、結果を見ただけではわかりません。SELECT に続く列名の指定において、SQL のキーワード AS を使って列に別名を与えれば、列の意味を明確にできます。

```
SELECT area_code.name AS area_name, n_hospital.num AS hospital_num
FROM area_code, n_hospital
WHERE area_code.code = n_hospital.code AND area_code.code = '01202';
```

```
area_name | hospital_num
-----+-----
函館市    |           204
(1 行)
```

4.2 相関名 (テーブルの別名)

テーブル名に、より短い別名 (相関名) を与えて、複数のテーブルを扱う SQL 文を短く書くことができます⁴。

相関名は FROM に続くテーブル名の後で AS に続き定義します⁵。定義した相関名は、SELECT に続く列名や WHERE 検索条件で使用できます。

次の例ではテーブル area_code に相関名 a を、テーブル n_hospital には相関名 h を与えています。

⁴SELECT 文では同一テーブル同士に結合 (自己結合) を施すこともできます。その場合、FROM の後ろのテーブル名に重複は許されないので、テーブルに相関名を定義して、異なるテーブル名で結合する必要があります。

⁵相関名のための AS は省略可能。

```
SELECT a.name, h.num
FROM area_code AS a, n_hospital AS h
WHERE a.code = h.code AND a.code = '01202';
```

```
name | num
-----+-----
函館市 | 204
(1 行)
```

5 問題

1. 前回資料の最後の問題

テーブル `postcode` から、列 `pub.auth.code` の左から 3 つ目の数字が 1 である行のみを表示しなさい。ただし、表示する列は `pub.auth.code`, `ken.kana`, `town.kana` の 3 列のみとし、結果が重複する行は出力しないこととする。

の出力を、都道府県名 (`ken.kana` 列) で昇順に並べなさい。同じ都道府県名の行については、さらに市区名 (`town.kana` 列) で昇順に並べなさい。

2. 次の各テーブルについて、各列のデータ型を確認しなさい。また、テーブルに含まれる列と行をすべて表示しなさい。

(a) テーブル `population`

列: `name` 地域名, `popul` 人口, `house` 世帯数

(b) テーブル `spring`

列: `name` 温泉名, `area` 地域コード

3. テーブル `population` から人口が 200000 以上の自治体の名前と人口を出力する SQL 文を実行しなさい。出力は次の通りとなる。

```
name | popul
-----+-----
函館市 | 265979
(1 行)
```

4. 次の出力を得るための SQL 文を実行しなさい。表示される行の順番は問わない。(以下の設問も同様)

```
name | code | name | popul | house
-----+-----+-----+-----+-----
鹿部町 | 01343 | 鹿部町 | 4226 | 1660
七飯町 | 01337 | 七飯町 | 28120 | 11141
森町 | 01345 | 森町 | 15946 | 6628
函館市 | 01202 | 函館市 | 265979 | 123950
北斗市 | 01236 | 北斗市 | 46390 | 18508
(5 行)
```

5. テーブル `spring` と `area_code` から、次の出力を得るための SQL 文を実行しなさい。

| name | location |
|--------|----------|
| 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 | 函館市 |
| せせらぎ温泉 | 北斗市 |
| 仁山温泉 | 七飯町 |
| 東大沼温泉郷 | 七飯町 |
| 鹿部温泉郷 | 鹿部町 |
| 濁川温泉郷 | 森町 |

(9 行)

6. テーブル `area_code` と `population` を使って、人口 (列 `popul`) が 30000 人以上の地域のコードと人口を表示しなさい。
7. 次の出力を得るための SQL 文を実行しなさい。 `area_code`, `population`, `n_hospital` の三つのテーブルを用いる。

| name | code | name | popul | house | code | num |
|------|-------|------|--------|--------|-------|-----|
| 函館市 | 01202 | 函館市 | 265979 | 123950 | 01202 | 204 |
| 北斗市 | 01236 | 北斗市 | 46390 | 18508 | 01236 | 23 |
| 七飯町 | 01337 | 七飯町 | 28120 | 11141 | 01337 | 18 |
| 鹿部町 | 01343 | 鹿部町 | 4226 | 1660 | 01343 | 2 |
| 森町 | 01345 | 森町 | 15946 | 6628 | 01345 | 9 |

(5 行)

8. 次の出力を得るための SQL 文を実行しなさい。ただし、 `area_code` の相関名として `a` を、 `population` の相関名として `p` を、 `n_hospital` の相関名として `h` を用いること。

| name | code | population | house | hospital |
|------|-------|------------|--------|----------|
| 函館市 | 01202 | 265979 | 123950 | 204 |
| 北斗市 | 01236 | 46390 | 18508 | 23 |
| 七飯町 | 01337 | 28120 | 11141 | 18 |
| 鹿部町 | 01343 | 4226 | 1660 | 2 |
| 森町 | 01345 | 15946 | 6628 | 9 |

(5 行)