東京ITスクール
TOKYO IT SCHOOL

集約と並べ替え

目次

1. テーブルを集約して検索する

1　集約関数

SQL でデータに対して何らかの操作や計算を行うには、関数を使用します。例えば、「テーブル全体の行数を数える」という計算を行うときは、COUNT という関数を使います。

SQL には沢山の集計用の関数が用意されていますが、まずは基本となる次の 5 つを覚えておきましょう。

COUNT　：テーブルのレコード数（行数）を数える
SUM　　：テーブルの数値列のデータを合計する
AVG　　：テーブルの数値列のデータを平均する
MAX　　：テーブルの任意の列のデータの最大値を求める
MIN　　：テーブルの任意の列のデータの最小値を求める

このような集計用の関数を「集約関数」や「集合関数」と呼びます。上で紹介した 5 つの関数は、以下のような形式で記述します。

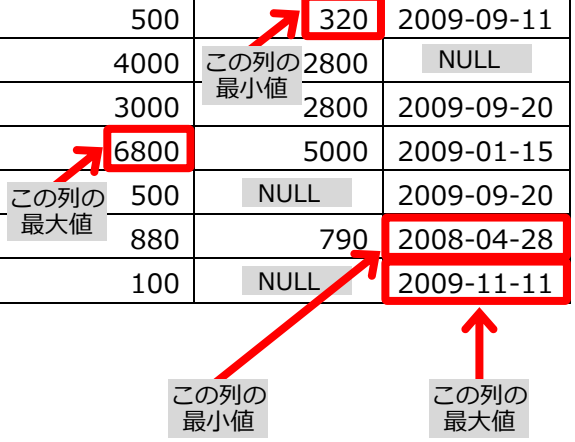
（構文）関数

```
関数名(列名)
```

関数の学習には演算子の章で使用した商品テーブル（Item）を利用します。

Item テーブル

item_id（商品 ID）	item_name（商品名）	category_id（商品分類）	sel_price（販売単価）	pur_price（仕入単価）	reg_date（登録日）
0001	シャツ	1	1000	500	2009-09-20
0002	ホッチキス	2	500	320	2009-09-11
0003	セーター	1	4000	2800	NULL
0004	包丁	3	3000	2800	2009-09-20
0005	フライパン	3	6800	5000	2009-01-15
0006	フォーク	3	500	NULL	2009-09-20
0007	スプーン	3	880	790	2008-04-28
0008	ボールペン	2	100	NULL	2009-11-11



2　テーブルの行数を数える

テーブルの行数を求める場合は、COUNT 関数を使用します。テーブル全体の行数を求めたい場合は、COUNT 関数の引数には、「全ての列」を意味する「*（アスタリスク）」を指定します。また、関数の引数に「*」を使用出来るのは、COUNT 関数のみとなります。

以下に Item テーブルの全行数を求める例を示します。

（サンプルコード）全行を数える

```
SELECT COUNT(*) FROM Item;
```

（実行結果）

```
COUNT(*)
----------
8
```

任意の列から NULL の行を除外して数えたい場合は、対象とする列を限定して引数に記述します。以下に「sel_price」列から NULL を除外した件数を求める例を示します。

（サンプルコード）任意の列から NULL を除いた行数を数える

```
SELECT COUNT(sel_price) FROM Item;
```

（実行結果）

```
COUNT(SEL_PRICE)
----------
6
```

3　合計を求める

任意の列の合計値を求める場合は SUM 関数を使用します。SUM 関数で指定出来る列は数値型に限られます。また、NULL 値が存在している場合、NULL 値は無視されて計算されます。

以下に「sel_price」列の合計を求める例を示します。

（サンプルコード）任意の列の合計を計算する

```
SELECT SUM(sel_price) FROM Item;
```

（実行結果）

```
SUM(SEL_PRICE)
----------
14290
```

4　平均値を求める

任意の列の平均値を求める場合は AVG 関数を使用します。AVG 関数で指定出来る列は数値型に限られます。また、NULL 値が存在している場合、NULL 値は無視されて計算されます。分母からも除外される為、十分注意をしましょう。

以下に「sel_price」列の平均を求める例を示します。

（サンプルコード）任意の列の平均を計算する

```
SELECT AVG(sel_price) FROM Item;
```

（実行結果）

```
AVG(SEL_PRICE)
------------------------
2381.66667
```

5　最大値・最小値を求める

任意の列の最大値または最小値を求める場合はそれぞれ MAX 関数、MIN 関数を使用します。これらの関数は文字型や日付型など数値型以外の列も指定出来ます。

以下に「sel_price」列の最大値と最小値を求める例を示します。

（サンプルコード）任意の列の最大値・最小値を計算する

```
SELECT MAX(sel_price), MIN(sel_price) FROM Item;
```

（実行結果）

```
MAX(SEL_PRICE)      MIN(SEL_PRICE)
----------------------- -------------------------
5000                    500
```

6　重複値を除外して集約関数を使う（DISTINCT キーワード）

重複値を除外して集約関数を使用する場合は、DISTINCT キーワードを用います。DISTINCT キーワードは引数として使用します。以下に構文を示します。

（構文）重複値を除外して集約関数を使う

```
関数名(DISTINCT 列名)
```

COUNT 関数を用いて「category_id」列の種類を数える例を以下に示します。

（サンプルコード）重複値を除外して集約関数を使う

```
SELECT COUNT(DISTINCT category_id) FROM Item;
```

（実行結果）

```
COUNT(DISTINCTCATEGORY_ID)
-------------------------------------------
3
```

このとき、DISTINCT は必ず引数として書く必要がある点に注意してください。なぜなら、「最初に「category_id」列の重複値を除外し、それからその結果の行数を数える」必要があるからです。もし、「SELECT DISTINCT COUNT(category_id) FROM Item;」のように引数の外に書いてしまうと、「最初に「category_id」列の行数を数え、それからその結果の重複値を除外する」ことになり、結果は「category_id」列の全行数（8）となります。

2. テーブルをグループに切り分ける

1　GROUP BY 句

これまで見てきた集約関数の使い方は、NULL を含む・含まない、重複値を除外する・除外しないという区別はありましたが、共通してテーブル全体を集約範囲としてきました。今度は、テーブルをいくつかのグループに切り分けて集約してみましょう。

グループ分けに用いるのが GROUP BY 句です。構文は次の通りです。

（構文）GROUP BY 句による集約

```
SELECT <列名 1>, <列名 2>, <列名 3>,……
  FROM <テーブル名>
GROUP BY <列名 1>, <列名 2>, <列名 3>,……;
```

まずは、「category_id」（商品分類）毎の行数（＝商品数）を数えてみましょう。以下に例を示します。

（サンプルコード）GROUP BY 句による集約

```
SELECT category_id, COUNT(*)
  FROM Item
GROUP BY category_id;
```

（実行結果）

```
CATEGORY_ID         COUNT(*)
----------------------- -----------------
1                   2
2                   2
3                   4
```

上の実行例のように、GROUP BY 句を使っていなかったときは、1 行だけだった結果が、今度は複数行に増えました。これは、GROUP BY 句なしの場合はテーブル全体を 1 つのグループと見なしていたのに対し、GROUP BY 句を使うことで複数のグループに切り分けられることになったためです。

なお、GROUP BY 句の位置にも厳密なルールがあって、必ず FROM 句の後ろ（WHERE 句があるならさらにその後ろ）におく必要があります。また、GROUP BY 句を使用する場合、SELECT 句に集約キー以外の列名を書くとことは出来ません。こちらも注意が必要です。

2　集約キーに NULL が含まれていた場合

集約キーに NULL が含まれる場合、それらは一括して「NULL」という 1 つのグループに分類されます。

3　WHERE 句を使った場合の GROUP BY の動作

GROUP BY 句を使う SELECT 文でも、WHERE 句は問題なく併用出来ます。句の記述順は先ほど説明した通りなので、構文は次のようになります。

（構文）WHERE 句と GROUP BY 句による集約

```
SELECT <列名 1>, <列名 2>, <列名 3>,……
  FROM <テーブル名>
  WHERE
GROUP BY <列名 1>, <列名 2>, <列名 3>,……;
```

このように WHERE 句をつけた集約を行なう場合、WHERE 句で指定した条件で先にレコードが絞り込まれてから集約が行われます。
以下に例文を示します。

（サンプルコード）WHERE 句と GROUP BY 句による集約

```
SELECT pur_price, COUNT(*)
  FROM Item
WHERE category_id = 1
GROUP BY pur_price;
```

この SELECT 文では、まず WHERE 句でレコードが絞り込まれるため、実際に集約対象になるレコードは次の表のように 2 行に絞られます。

WHERE 句で絞り込んだ結果

item_id（商品 ID）	item_name（商品名）	category_id（商品分類）	sel_price（販売単価）	pur_price（仕入単価）	reg_date（登録日）
1	シャツ	0001	1000	500	2009-09-20
1	セーター	0003	4000	2800	

この 2 つのレコードについて仕入単価でグループ化するため、実行結果は次のようになります。

（実行結果）

```
PUR_PRICE           COUNT(*)
---------- ----------
500                 1
2800                1
```

4　集約関数と GROUP BY 句にまつわる注意点

ここまで、集約関数と GROUP BY 句の基本的な使い方を学びました。これらはいずれも使用頻度が高いですが、SQL を書く際に間違えやすい点や注意しておくべき点がいくつかあります。

■注意点①---------- GROUP BY 句に列の別名を書いてしまう

SELECT 句に含めた項目には、「AS」というキーワードを使うことで、表示用の別名をつけることが出来ます。しかし、GROUP BY 句でこの別名を使うことは出来ません。

理由は、DBMS 内部で SQL 文が実行される順序において、SELECT 句が GROUP BY 句よりも後に実行されるからです。そのため、GROUP BY 句の時点では SELECT 句でつけた別名を、DBMS はまだ知らないのです。

■注意点②---------- WHERE 句に集約関数を書いてしまう

WHERE 句には集約関数を用いることは出来ません。理由は「WHERE 句を使った場合の GROUP BY の動作」でもあるように、GROUP BY 句や SELECT 句よりも先に WHERE 句が実行されるからです。

具体例として、「category_id」列（商品分類）でグルーピングして行数を数える SQL を見てみましょう。以下に SELECT 文を示します。

（サンプルコード）商品分類ごとに行数を数える

```
SELECT category_id, COUNT(*)
  FROM Item
GROUP BY category_id;
```

（実行結果）

```
CATEGORY_ID          COUNT(*)
-------------------- --------------------
1                    2
2                    2
3                    4
```

この結果を見て、今度は「この数えた行数が、丁度 2 行のグループだけ選択したい」と思い、以下のような SQL 文を記述する場合があります。こちらはエラーとなりますので注意してください。

（サンプルコード）WHERE 句に集約関数を記述

```
SELECT category_id, COUNT(*)
  FROM Item
  WHERE COUNT(*) = 2
GROUP BY category_id;
```

（実行結果）

```
次のコマンド行の開始中にエラーが発生しました ：1 -
SELECT category_id, COUNT(*)
   FROM Item
   WHERE COUNT(*) = 2
GROUP BY category_id
コマンド行 ：3 列 ：9 でのエラー
エラー・レポート -
SQL エラー: ORA-00934: ここではグループ関数は使用出来ません。
00934. 00000 -  "group function is not allowed here"
*Cause:
*Action:
```

実は、COUNT など集約関数を書くことの出来る場所は、SELECT 句と後述の HAVING 句と ORDER BY 句だけなのです。そしてこの新しく登場した HAVING 句こそが、「2 行のグループだけ選択する」のようなグループに対する条件を指定するための便利な道具です。

3. 集約した結果に条件を指定する

1　HAVING 句

前節で学んだ GROUP BY 句によって、元のテーブルをグループ分けして結果を得ることが出来るようになりました。ここでは、更にそのグループに対して条件を指定して選択する方法を考えます。

条件指定といえば、WHERE 句が真っ先に思い浮かぶかもしれません。しかし、WHERE 句はあくまで「レコード（行）」に対してのみしか条件を指定出来ないため、グループに対する条件指定（例えば、「含まれる行数が 2 行」や「平均値が 500」など）には使えないという制限があります。従って、そういう集合に対する条件を指定する句が、別に必要となります。それが HAVING 句です。

HAVING 句の構文は次の通りです。

（構文）HAVING 句

```
SELECT <列名 1>, <列名 2>, <列名 3>,……
  FROM <テーブル名>
  GROUP BY <列名 1>, <列名 2>, <列名 3>,……
HAVING <グループの値に対する条件>;
```

HAVING 句を記述する位置は、GROUP BY 句の後ろである必要があります。DBMS 内部での実行順序も、GROUP BY 句の後になります。

▶ HAVING 句を使用するときの SELECT 文の記述順序
SELECT → FROM → WHERE → GROUP BY → HAVING

商品分類で集約したグループに対して、「含まれる行数が 2 行」という条件を指定する例を以下に示します。

（サンプルコード）HAVING 句

```
SELECT category_id, COUNT (*)
  FROM Item
  GROUP BY category_id
HAVING COUNT(*) = 2;
```

（実行結果）

```
CATEGORY_ID          COUNT(*)
-------------------- --------------------
1                    2
2                    2
```

また、HAVING 句も、GROUP BY 句を使ったときの SELECT 句と同様に、GROUP BY 句で指定した列名（つまり集約キー）以外を列名として指定することは出来ません。

2　HAVING 句よりも WHERE 句に書いたほうがよい条件

中には、HAVING 句にも WHERE 句にも書ける条件というものが存在します。それは、「集約キーに対する条件」です。元のテーブルの列のうち、集約キーとして使っているものは、HAVING 句にも書くことが出来ます。従って、以下の SELECT 文は正しい構文です。

（サンプルコード）条件を HAVING 句に書いた場合

```
SELECT category_id, COUNT (*)
   FROM Item
   GROUP BY category_id
HAVING category_id = 1;
```

（実行結果）

```
CATEGORY_ID          COUNT(*)
-------------------- --------------------
1                    2
```

この SELECT 文は、以下のように書いた場合と同じ結果を返します。

（サンプルコード）条件を WHERE 句に書いた場合

```
SELECT category_id, COUNT (*)
   FROM Item
WHERE category_id = 1
   GROUP BY category_id;
```

（実行結果）

```
CATEGORY_ID          COUNT(*)
-------------------- --------------------
1                    2
```

条件を書く場所が WHERE 句か HAVING 句かの違いだけで、条件の内容は同じで、返す結果も同じです。従って、どちらの書き方をしてもよいではないか、と思うかもしれません。

選択される結果だけを見るなら、その通りですが、こういう集約キーに対する条件は WHERE 句に書くべきだと言えるでしょう。理由は 2 つあります。

まず 1 つは、WHERE 句と HAVING 句の役割の違いという、根本的なものです。HAVING 句というものは「グループ」に対する条件を指定するものです。従って、単なる「行」に対する条件は、WHERE 句で書くようにしたほうが、互いの機能をはっきりさせることが出来て、読みやすいコードになります。

WHERE 句＝行に対する条件指定
HAVING 句＝グループに対する条件指定

もう 1 つの理由は、WHERE 句と HAVING 句の実行速度の違いです。一般的に、同じ結果が得られるにせよ、HAVING 句よりは WHERE 句に条件を記述するほうが、処理速度が速く、結果が返ってくる時間も短くなります。

4. 検索結果を並べ替える

1　ORDER BY 句

SELECT 文の結果を並び替える場合は、ORDER BY 句を使用します。構文は以下の通りです。

（構文）ORDER BY

```
ORDER BY 列名 順序（昇順または降順）
```

ORDER BY 句は必ず SELECT 文の最後尾に書く必要があります。また、ORDER BY 句に書く列名を「ソートキー」と呼びます。
他の句との順序関係を表わすと次のようになります。

▶ 句の記述順序
1.SELECT 句 → 2.FROM 句 → 3.WHERE 句 → 4.GROUP BY 句 → 5.HAVING 句 → ORDER BY 句

なお、ORDER BY 句は、行の順番を指定したいと思わなければ、別に書かなくてもかまいません。また、順序の指定は昇順であれば、「ASC」、降順であれば「DESC」を記述します。昇順の場合は省略可能です。
以下に販売単価の高い順、つまり降順に並べる例を示します。

（サンプルコード）ORDER BY

```
SELECT *
    FROM Item
ORDER BY sel_price DESC;
```

（実行結果）

```
ITEM_ID  ITEM_NAME            CATEGORY_ID          SEL_PRICE            PUR_PRICE            REG_DATE
---------- -------------------- -------------------- -------------------- -------------------- --------
0006       フォーク             3                                         2800                 09-09-20
0008       ボールペン           2                                         2800                 09-11-11
0005       フライパン           3                    5000                 2800                 09-01-15
0003       セーター             1                    4000                 2800
0004       包丁                 3                    3000                 2800                 09-09-20
0001       シャツ               1                    1000                 500                  09-09-20
0007       スプーン             3                    790                  2800                 08-04-28
0002       ホッチキス           2                    500                  320                  09-09-11

8 行選択されました
```

2　複数のソートキーを指定する

「同順位」の行について更に細かく並び順を指定したい場合は、もう 1 つソートキーを追加する必要があります。ソートキーはカンマ区切りで複数指定することが出来ます。
販売単価の降順、商品 ID の昇順で並び替えを行う例を以下に示します。

（サンプルコード）複数のソートキーを指定した ORDER BY

```
SELECT *
    FROM Item
ORDER BY sel_price DESC, item_id ASC;
```

（実行結果）

ITEM_ID	ITEM_NAME	CATEGORY_ID	SEL_PRICE	PUR_PRICE	REG_DATE
0006	フォーク	3		2800	09-09-20
0008	ボールペン	2		2800	09-11-11
0005	フライパン	3	5000	2800	09-01-15
0003	セーター	1	4000	2800	
0004	包丁	3	3000	2800	09-09-20
0001	シャツ	1	1000	500	09-09-20
0007	スプーン	3	790	2800	08-04-28
0002	ホッチキス	2	500	320	09-09-11

8 行選択されました

このように、ORDER BY 句には、複数のソートキーを指定することが可能です。左側のキーから優先的に使用され、そのキーで同じ値が存在した場合に、右のキーが参照される、というルールです。もちろん、3 列以上のソートキーを使うことも可能です。

3　NULL の順番

NULL を含む列をソートキーにした場合、NULL は先頭または末尾にまとめて表示されます。先頭に来るか末尾に来るかは、特に決まっていません。中には、先頭か末尾かを指定することの出来る DBMS もあります。Oracle の場合、NULL はデフォルトの昇順ソートでは最後、降順ソートでは最初に並び替えられます。また、ORDER BY 句のオプション指定で「NULLS FIRST」を指定すると NULL が先頭に、「NULLS LAST」を指定すると NULL が最後に表示されます。

4　ソートキーに表示用の別名を使う

GROUP BY 句には、SELECT 句でつけた列の別名は使うことが許されていませんが、ORDER BY 句ではそれが許されています。したがって、以下のような SELECT 文は、エラーにはならず、正しく実行出来ます。

（サンプルコード）別名を使用した ORDER BY

```
SELECT item_id AS id, item_name
    FROM Item
ORDER BY id ASC;
```

（実行結果）

```
ID         ITEM_NAME
---------- --------------------
0001       シャツ
0002       ホッチキス
0003       セーター
0004       包丁
0005       フライパン
0006       フォーク
0007       スプーン
0008       ボールペン

8 行選択されました
```

GROUP BY 句では使えない別名が、ORDER BY 句では使える理由は、DBMS 内部で SQL 文が実行される順序に隠されています。SELECT 文の実行順序は、句単位で見ると次のようになります。

▶ SELECT 句の実行順序

FROM → WHERE → GROUP BY → HAVING → SELECT → ORDER BY

5　ORDER BY 句に使える列

ORDER BY 句には、テーブルに存在する列であれば、SELECT 句に含まれていない列でも指定出来ます。集約関数も使うことが出来ます。

また、ORDER BY 句では、SELECT 句に含まれる列を参照する列番号を使うことが出来ます。列番号とは、SELECT 句で指定した列を左から 1,2,3.....と順番を割り振った番号です。列番号で指定する書き方は、列名を書かなくてもよいという手軽さもあって非常に便利なのですが、次の 2 つの理由から使うべきではありません。

まず 1 つ目の理由は、コードが読みにくいことです。列番号を使うと、ORDER BY 句を見ただけではどんな列をソートキーにしているか分からず、SELECT 句のリストを先頭から数えなければなりません。SELECT 句にたくさんの列を含めることもありますし、SELECT 句と ORDER BY 句の間に大きな WHERE 句や HAVING 句が挟まって、目で追うことが大変な場合もあります。

そして 2 つ目の理由は、もっと根本的な問題です。実は、この順番項目の機能は、SQL-92 において、「将来削除されるべき機能」に挙げられました。従って、現在は問題なくても、将来、DBMS のバージョンアップを行なった際に、これまで動いていた SQL が突如エラーになるという厄介な問題を引き起こす可能性があります。その場限りの使い捨ての SQL ならまだしも、システムに組み込み SQL でこの機能を使うことは、避けたほうがよいでしょう。