第８章　SQLで高度な処理を行う

　　８－１ ウィンドウ関数

ウィンドウ関数とは・・・OLAP関数とも呼ぶリアルタイムでデータの分析を行う処理。

RANK関数・・・レコードのランキングを算出する関数

|  |  |  |  |
| --- | --- | --- | --- |
| 商品名 | 商品分類 | 単価 |  |
| フォーク | キッチン用品 | 500 | 1 |
| おろしがね | キッチン用品 | 880 | 2 |
| 包丁 | キッチン用品 | 3000 | 3 |
| 圧力鍋 | キッチン用品 | 6800 | 4 |
| ボールペン | 事務用品 | 100 | 1 |
| 穴あけパンチ | 事務用品 | 500 | 2 |
| Tシャツ | 衣服 | 2000 | 1 |
| カッターシャツ | 衣服 | 3000 | 2 |

例

SELECT shohin\_mei,shohin\_bunrui,hanbai\_tanka,

RANK () OVER (PARTITION BY shohin\_bunrui

　　　　　 ORDER BY hanbai\_tanka)AS ranking

FROM shohin;

RANK () OVER のところで商品分類を販売単価順にランキングにするという列ができた。

PARTITION BY ・・・順位をつける対象の範囲、これによっ

　　て区切られた部分集合をウィンドウと呼ぶ

ORDER　BY・・・ここではどんな順序でランキングをつけるかの指定

　※PARTITION　BYがない場合はすべてを含めてのランキングの形になる。

ROW＿NUMBER関数・・・指定したものに連番をつける関数、数を数えるのに便利

DENSE\_RANK関数・・・順位をつけるが同じ順位のものの後でも数字が飛ばずに表示される

例

SELECT shohin\_mei,shohin\_bunrui,hanbai\_tanka,

RANK () OVER (ORDER BY hanbai\_tanka) AS ranking,

DENSE\_RANK () OVER(ORDER BY hanbai\_tanka) AS dense\_ranking,

ROW\_NUMBER () OVER(ORDER BY hanbai\_tanka) AS row\_num

FROM shohin;

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 商品名 | 商品分類 | 販売単価 | ランキング | デンスランク | ロウナンバー |
| ボールペン | 事務用品 | 100 | 1 | 1 | 1 |
| 穴あけパンチ | 事務用品 | 500 | 2 | 2 | 2 |
| フォーク | キッチン用品 | 500 | 2 | 2 | 3 |
| おろしがね | キッチン用品 | 880 | 4 | 3 | 4 |
| Tシャツ | 衣服 | 2000 | 5 | 4 | 5 |
| カッターシャツ | 衣服 | 3000 | 6 | 5 | 6 |
| 包丁 | キッチン用品 | 3000 | 6 | 5 | 7 |
| 圧力鍋 | キッチン用品 | 6800 | 8 | 6 | 8 |

カレントレコード・・・今自分のいる位置を示したもの。

集約関数をウィンドウ関数としてつかう・・・SUMやAVGなどの集約関数をウィンドウ関数とあわせ

　　　　ることで途中経過を見ることができる。

例

|  |  |  |  |
| --- | --- | --- | --- |
| 0001 | Tシャツ | 2000 | 2000 |
| 0002 | 穴あけパンチ | 500 | 2500 |
| 0003 | カッターシャツ | 3000 | 5500 |
| 0004 | 包丁 | 3000 | 8500 |
| 0005 | 圧力鍋 | 6800 | 15300 |
| 0006 | フォーク | 500 | 15800 |
| 0007 | おろしがね | 880 | 16680 |
| 0008 | ボールペン | 100 | 16780 |

SELECT shohin\_id,shohin\_mei,hanbai\_tanka,

SUM(hanbai\_tanka) OVER(ORDER BY shohin\_id)

FROM shohin;

商品IDをランキング形式にして表示して、

上から順にどんどんSUM（合計値の計算）

されている列を作ることができる。

移動平均を算出する・・・下の関数を使って～行前までの平均を出すやり方

|  |  |  |  |
| --- | --- | --- | --- |
| 0001 | Tシャツ | 2000 | 2000.0000 |
| 0002 | 穴あけパンチ | 500 | 1250.0000 |
| 0003 | カッターシャツ | 3000 | 1833.3333 |
| 0004 | 包丁 | 3000 | 2166.6667 |
| 0005 | 圧力鍋 | 6800 | 4266.6667 |
| 0006 | フォーク | 500 | 3433.3333 |
| 0007 | おろしがね | 880 | 2726.6667 |
| 0008 | ボールペン | 100 | 493.3333 |

例

AVG(hanbai\_tanka) OVER(ORDER BY shohin\_id

ROWS 2 PRECEDING) AS current\_sum

FROM shohin;

これで平均値の途中経過も表示されるものが

作られた。

ROWS・・・　～行というフレームの指定ができる

PRECEDING・・・～前までという意味でROWSと合わせて使う。

ROWS 2 PRECEDINGは自分より２行前までの指定となる。

上のものでは自分より前のものだが次のものを使えば上下のものを計算対象にできる。

例

SELECT shohin\_id,shohin\_mei,hanbai\_tanka,

AVG(hanbai\_tanka) OVER (ORDER BY shohin\_id

ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING) AS moving\_avg

FROM shohin;

これで自分の上１つとした１つのものを含めた３つの行の平均値を出すことができるようになる。

２つのORDER　BY・・・OVERの中のORDER　BY　と最後のORDER BYでは意味が違い、最初の

　　ORDER BYはランキングを何を基準にするか決めるORDER　BY　でその後

　　のORDER BYはそのものの並び方を決めるORDER BYである。

例

SELECT shohin\_mei,shohin\_bunrui,hanbai\_tanka,

RANK () OVER (ORDER BY hanbai\_tanka) AS ranking

FROM shohin

ORDER BY ranking;

上は販売単価のランキングと決めるオーダーバイ、下のものはランキング順に並べるという表示。

８－２　GROUPING演算子

GROUPING・・・集約単位の違う集約結果を求めることができる。

1. ROLLUP・・・合計値を求めることができる。

②CUBE・・・

　　③GROUPUP・・・

まずおさらいとして。

|  |  |
| --- | --- |
| 衣服 | 5000 |
| 事務用品 | 600 |
| キッチン用品 | 11180 |

SELECT shohin\_bunrui,SUM(hanbai\_tanka)

FROM shohin

GROUP BY shohin\_bunrui;

これは商品分類グループごとの合計値を算出するものです。

この３つの商品分類の合計値の合計を求める場合はまた違う文が必要になります。

UNION ALL・・・グループバイなどで分けたものをまとめたものを表示したい場合に使うコマンド。

|  |  |
| --- | --- |
| 合計 | 16780 |
| 衣服 | 5000 |
| 事務用品 | 600 |
| キッチン用品 | 11180 |

例

SELECT '合計' AS shohin\_bunrui,SUM(hanbai\_tanka)

FROM shohin

UNION ALL

SELECT shohin\_bunrui,SUM(hanbai\_tanka)

FROM shohin

GROUP BY shohin\_bunrui;

こうすることで合計値を上に表示することができる。

しかし、このままだと同じようなことを書いてて面倒くさいので書き換えたものが次のも

1. ROLLUP・・・合計と承継を一度に求めることができる関数

|  |  |
| --- | --- |
| キッチン用品 | 11180 |
| 事務用品 | 600 |
| 衣服 | 5000 |
|  | 16780 |

例

SELECT shohin\_bunrui,SUM(hanbai\_tanka) AS sum\_tanka

FROM shohin

GROUP BY shohin\_bunrui WITH ROLLUP;

このように短縮して表示できる。

またROLLUPでは複数のものを集約キーに追加することもできる。

|  |  |  |
| --- | --- | --- |
| キッチン用品 | 2008-04-28 | 880 |
| キッチン用品 | 2009-01-15 | 6800 |
| キッチン用品 | 2009-09-20 | 3500 |
| キッチン用品 |  | 11180 |
| 事務用品 | 2009-09-11 | 500 |
| 事務用品 | 2009-11-11 | 100 |
| 事務用品 |  | 600 |
| 衣服 |  | 3000 |
| 衣服 | 2009-09-20 | 2000 |
| 衣服 |  | 5000 |
|  |  | 16780 |

例

SELECT shohin\_bunrui,torokubi,SUM(hanbai\_tanka)

FROM shohin

GROUP BY shohin\_bunrui,torokubi　WITH ROLLUP;

これで商品分類ごとの合計値が出て、登録日も表示できる、衣服の一つ目のＮＵＬＬはもともとＮＵＬＬだからＮＵＬＬなだけほかは合計値には登録日がないのでＮＵＬＬ。

偽物のＮＵＬＬを見分けろ

先ほどの表の衣服にＮＵＬＬがありますが、これと合計値のところのＮＵＬＬの見分けをつくようにする方法です。

|  |  |  |
| --- | --- | --- |
| 分類 | 登録日 | 単価合計 |
| 0 | 0 | 880 |
| 0 | 0 | 6800 |
| 0 | 0 | 3500 |
| 0 | 1 | 11180 |
| 0 | 0 | 500 |
| 0 | 0 | 100 |
| 0 | 1 | 600 |
| 0 | 0 | 3000 |
| 0 | 0 | 2000 |
| 0 | 1 | 5000 |
| 1 | 1 | 16780 |

例

SELECT GROUPING(shohin\_bunrui)

GROUPING(torokubi),SUM(hanbai\_tanka)

FROM shohin

GROUP BY shohin\_bunrui,torokubi WITH ROLLUP;

このように表示される。

ＮＵＬＬなら１を、ＮＵＬＬではないなら０を返却する文。

でもこれだと少し見分けがつきにくいのでそれを見やすくしたもの

が次の文になります。

例

SELECT CASE WHEN GROUPING(shohin\_bunrui) = 1 　グルーピングしたとき結果が１の

THEN '商品分類　合計' 　部分は商品分類　合計と表示

ELSE shohin\_bunrui END,　そうでないところは商品分類の表示

CASE WHEN GROUPING(torokubi) = 1 グルーピングしたときの結果が1の

THEN '登録日　合計' 部分は登録日　合計と表示

ELSE CAST(torokubi AS CHAR(16)) END,　それ以外は登録日の表示

|  |  |  |
| --- | --- | --- |
| キッチン用品 | 2008-04-28 | 880 |
| キッチン用品 | 2009-01-15 | 6800 |
| キッチン用品 | 2009-09-20 | 3500 |
| キッチン用品 | 登録日　合計 | 11180 |
| 事務用品 | 2009-09-11 | 500 |
| 事務用品 | 2009-11-11 | 100 |
| 事務用品 | 登録日　合計 | 600 |
| 衣服 |  | 3000 |
| 衣服 | 2009-09-20 | 2000 |
| 衣服 | 登録日　合計 | 5000 |
| 商品分類　合計 | 登録日　合計 | 16780 |

SUM(hanbai\_tanka) AS sum\_tanka

FROM shohin

GROUP BY shohin\_bunrui,torokubi WITH ROLLUP;

このようにＲＯＬＬＵＰによって生まれたＮＵＬＬは

文字が入れられて、もともとＮＵＬＬだったものはそのままの表示がされるようになりました。