３章　集約と並べ替え

　　　３－１　テーブルを集約して検索する

　　　　　テーブルの行数を数える

　　　　　・・・テーブルの行数を使うときはCOUNT関数を使う。

COUNT関数・・・そのものの数を数える関数。（NULLはカウントされない）

例 　　　　結果 shohinテーブルの中の行数

|  |
| --- |
| COUNT |
| 8 |

SELECT　COUNT（＊）

　FROM　shohin；

　合計を求めるとき

・・・SUM関数をしようして合計を求める

SUM関数・・・そのものの合計を調べる関数。（NULLは飛ばして計算される）

例 結果 shohinテーブルの販売単価の合計

|  |
| --- |
| SUM |
| 16780 |

SELECT SUM(hanbai\_tanka)

FROM shohin;

平均値を求めるとき

・・・AVG関数を使用して平均値を求める

AVG関数・・・そのものの平均値を求める関数。（NULLは計算から除外）

例 　　　　結果 shohinテーブルの販売単価の平均

|  |
| --- |
| AVG |
| 2097.500 |

SELECT AVG

(hanbai\_tanka)

FROM shohin;

最大値・最小値を求める

・・・MAX関数、MIN関数を使う

MAX関数・・・そのものの最大値を調べる関数。（数字以外でも使える）

MIN関数・・・そのものの最小値を調べる関数。（数字以外でも使える）

例 　結果 shohinテーブルの販売単価の最大値

|  |
| --- |
| MAX |
| 6800 |

SELECT max(hanbai\_tanka)

FROM shohin;

重複地を除外して集約関数を使う

・・・DISTINCTを使う

DISTINCTキーワード・・・カウント関数などの()の中の先頭に入れて重複してるものを除外できる 例 　　結果 　　　　カウントは8だが重複を除外

|  |
| --- |
| COUNT |
| 3 |

SELECT COUNT(DISTINCT shohin\_bunrui)　　　　 するので結果は３になる。

FROM shohin;

　　　３－２　テーブルをグループに分ける

GROUP　BY句　・・・　～によってグループ分けするという意味。商品分類ごとや登録日ごと

　　　というように切り分けることができる。

　　　指定する列のことを集約キー、グループ化列と呼ぶ

|  |  |
| --- | --- |
| Shohin\_bunrui | count |
| 衣服 | 2 |
| 事務用品 | 2 |
| キッチン用品 | 4 |

例 結果

SELECT shohin\_bunrui,COUNT(\*)

FROM shohin

GROUP BY shohin\_bunrui;

商品分類というグループがつくられ、その中の数をカウントされた。

集約キーにNULLが含まれていた場合

・・・NULLがあった場合はそのNULLをまた、１グループととらえ表示する

|  |  |
| --- | --- |
| Shire\_tanka | count |
| 500 | 1 |
| 320 | 1 |
| 2800 | 2 |
| 5000 | 1 |
|  | 2 |
| 790 | 1 |

例 　　結果

SELECT shiire\_tanka,count(\*)

FROM shohin

GROUP BY shiire\_tanka;

今回は仕入単価のNULLが二つあるので

結果NULLのところにカウントが２つある

WHEREを使った場合のGROUP BYの動作

・・・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グループ分けされる。

例 結果

|  |  |
| --- | --- |
| Shire\_tanka | COUNT |
| 500 | 1 |
| 2800 | 1 |

SELECT shiire\_tanka,COUNT(\*)

FROM shohin

WHERE shohin\_bunrui = '衣服'

GROUP BY shiire\_tanka;

まずWHEREによって衣服のものがピックアップされ、そのあとにGROUP BYで仕入単価の

部分をグループ分けしている。

GROUP BY　のよくあるミス

1. SELECT句に余計な列を書いてしまう。

・・・SELECT句に使えるのは、定数、集約関数、GROUP BYで指定した列名のみ

例

SELECT shohin\_mei,shiire\_tanka,COUNT(\*) 単価のグループを作り単価が

FROM shohin 同じものをカウントしたが、商品名

GROUP BY shiire\_tanka; はカウントできないので同じものが二つ

あることになってしまう。

1. GEOUP BY句に列の別名を書いてしまう。

・・・AS　を使うことで別名をつけることができるがGROUP BYのところで別名を使うことはできない。

例

SELECT shohin\_bunrui AS sb,COUNT(\*) 　　　　基本DBMSはプログラムからは下から読ま

FROM shohin れるのでｓｂが何かわからなく、エラーが出る。

GROUP BY sb;

1. GROUP BY ha結果の順序をソートする

・・・グループの並び方はランダムなのでソートするためにはそのために指定をセレクト

文でするひつようがある。

1. WHEREに集約関数を書いてしまう。

・・・条件を指定するときはWHEREを使うのでそこでみようとしてしまう。

例

正しい 間違い

SELECT shohin\_bunrui,COUNT(\*) SELECT shohin\_bunrui,COUNT(\*)

FROM shohin FROM　shohin

GROUP BY shohin\_bunrui; WHERE COUNT(\*) = 2

GROUP BY shohin\_bunrui;

左のもので出た２のカウントのみを表示したいという意味で右のものを作ったがあくまでWHERE

が使えるのは一つ一つのレコードに向けてのみなのでこの分ではエラーが出てしまう。

３－３　集約した結果に条件を指定する

　HAVING句

・・・GROUP BYでできるグループにさらに条件を付けてグループ分けができる。

1. 結果

|  |  |
| --- | --- |
| Shohin\_bunrui | COUNT |
| 衣服 | 2 |
| 事務用品 | 2 |

SELECT shohin\_bunrui,COUNT(\*)

FROM shohin

GROUP BY shohin\_bunrui

HAVING COUNT(\*) = 2;

商品分類の中のカウントが２のもののみの表示ができる。

1. 結果

|  |  |
| --- | --- |
| Shohin\_bunrui | COUNT |
| キッチン用品 | 2795.0000 |

SELECT shohin\_bunrui,AVG(hanbai\_tanka)

FROM shohin

GROUP BY shohin\_bunrui

HAVING AVG(hanbai\_tanka) >2500;

商品分類の平均値を出し、その中でも平均値が２５００を超えたものだけを表示。

※HAVINGはGROUP BYで使用された列のものしか使えない。

HAVINGを使うかWHEREを使うか

　　どちらも条件を付けて検索するものだがその場合場合によって使い分けられるとよい。

例

SELECT shohin\_bunrui,COUNT(\*)　　　SELECT shohin\_bunrui,COUNT(\*)

FROM shohin 　　　FROM shohin

GROUP BY shohin\_bunrui 　　　GROUP BY shohin\_bunrui

HAVING shohin\_bunrui = '衣服' 　　　HAVING shohin\_bunrui = '衣服

検索結果は同じだが、WHEREは行に対する条件指定、HAVINGはグループに対する条件指定

なのでWHEREの方が早く済む。

３－４　検索結果を並べる

ORDER BY句

・・・明示的に行の順序を指定できる。

まず基本に戻るが

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

|  |  |  |  |
| --- | --- | --- | --- |
| Shohin\_id | Shohin\_mei | Hanbai\_tanka | Shire\_tanka |
| 0001 | Tシャツ | 1000 | 500 |
| 0002 | 穴あけパンチ | 500 | 320 |
| 0003 | カッターシャツ | 4000 | 2800 |
| 0004 | 包丁 | 3000 | 2800 |
| 0005 | 圧力鍋 | 6800 | 5000 |
| 0006 | フォーク | 500 |  |
| 0007 | おろしがね | 880 | 790 |
| 0008 | ボールペン | 100 |  |
|  |  |  |  |

FROM shohin;

'これの結果はこうなる。

これにORDER BY をつけることにより自分の思い通りの並び順に変える

ことができる。

例

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

FROM shohin

|  |  |  |  |
| --- | --- | --- | --- |
| Shohin\_id | Shohin\_mei | Hanbai\_tanka | Shire\_tanka |
| 0008 | ボールペン | 100 |  |
| 0002 | 穴あけパンチ | 500 | 320 |
| 0006 | フォーク | 500 |  |
| 0007 | おろしがね | 880 | 790 |
| 0001 | Tシャツ | 1000 | 500 |
| 0004 | 包丁 | 3000 | 2800 |
| 0003 | カッターシャツ | 4000 | 2800 |
| 0005 | 圧力鍋 | 6800 | 5000 |

ORDER BY hanbai\_tanka;

ORDER　BY　hanbai\_tankaをつけることにより販売単価の昇順に並べることができました。

逆に降順に並べたいときは

ORDER　BY　hanbai\_tanka　DESC;

というように後ろにDESCをつける。

　　　　複数のソートキーの指定

ORDER　BYで複数指定をすることでより細かい並び方の設定ができる。

例

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

|  |  |  |  |
| --- | --- | --- | --- |
| 0008 | ボールペン | 100 |  |
| 0002 | 穴あけパンチ | 500 | 320 |
| 0006 | フォーク | 500 |  |
| 0007 | おろしがね | 880 | 790 |
| 0001 | Tシャツ | 1000 | 500 |
| 0004 | 包丁 | 3000 | 2800 |
| 0003 | カッターシャツ | 4000 | 2800 |
| 0005 | 圧力鍋 | 6800 | 5000 |

FROM shohin

ORDER BY hanbai\_tanka,shohin\_id;

まず先にある販売単価が昇順に並べられ、販売単価が同じものがあったらその中で後ろの商品IDでのソートが

される。

　　　　NULLの順番

途中にNULLがあると見ずらい時があるがそういう時はNULLを含むところでORDER　BYをすれば綺麗になる。

例

|  |  |  |  |
| --- | --- | --- | --- |
| 0006 | フォーク | 500 |  |
| 0008 | ボールペン | 100 |  |
| 0002 | 穴あけパンチ | 500 | 320 |
| 0001 | Tシャツ | 1000 | 500 |
| 0007 | おろしがね | 880 | 790 |
| 0003 | カッターシャツ | 4000 | 2800 |
| 0004 | 包丁 | 3000 | 2800 |
| 0005 | 圧力鍋 | 6800 | 5000 |

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

FROM shohin

ORDER BY shiire\_tanka;

NULLのものをまとめることができる

　※NULLの付くものが上に行くか下に行くかはランダム

なのでしっかり決めたいならDESCなどをつけて定義する。

　　　　ORDER　BYの便利な点

・ソートキーの表示用の別名を使える・・・セレクト句で一つ一つにASをつけることでできる。

・集約関数も使える・・・ORDER BY　COUNT(＊)とすることもできる。←カウントの昇順並び

・セレクト句に含まれてない列もORDER BYで使うことができる

・列番号で指定ができる。

　　　　列番号・・・セレクト句で決めた列を左から１.２．３.と割り振った番号。

例

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

FROM shohin

ORDER BY shiire\_tanka　DESC　shohin\_id;

これを列番号で表せる。

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

FROM shohin

ORDER BY 3　DESC 1;

というように省略できるが、基本的に見にくくなるのと、将来的になくなる可能性が高いので

おすすめはしない。