4-4-5.SELECT文②

集計関数

はじめに

DBにため込んだデータを登録されたままの状態で抽出するだけでなく、様々な加工を施すことができます。
そのひとつが集計です。ここでは集計するためのSQL文を学んで行きましょう！

Step1 : 集計関数

SQLには、いくつか関数が用意されています。
基本的な集計関数と内容を表にしました。

集計関数	内容
COUNT()	件数を数えて返却する
SUM()	合計を算出して返却する
MAX()	最大値を探索して返却する
MIN()	最小値を探索して返却する
AVG()	平均値を算出して返却する

COUNT()

COUNT() は抽出された件数を出力します。

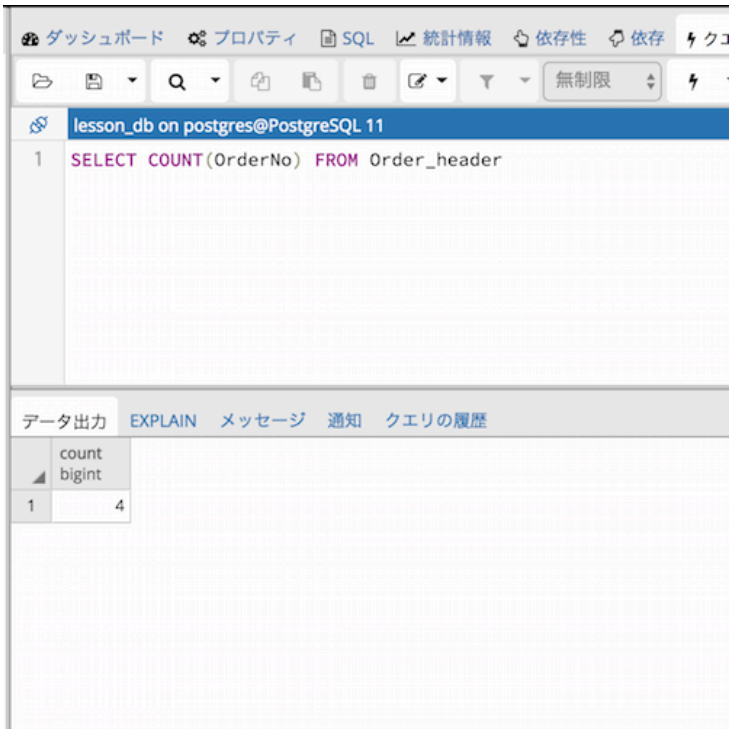
```
SELECT COUNT(列名) FROM テーブル名
```

列名はアスタリスクを使用しても構いません。
ただアスタリスクを使用すると全列が検索対象になるため、列名を指定するよりも処理時間がかかります。
また、NULLが入る可能性がある列を指定した場合、 **NULL** は件数にカウントされません。
全レコード件数を知りたい場合は主キーをカウント対象列としましょう。
WHERE句で条件をつければ対象レコード件数を知ることができます。

それでは現在登録されている注文件数をカウントしてみましょう。

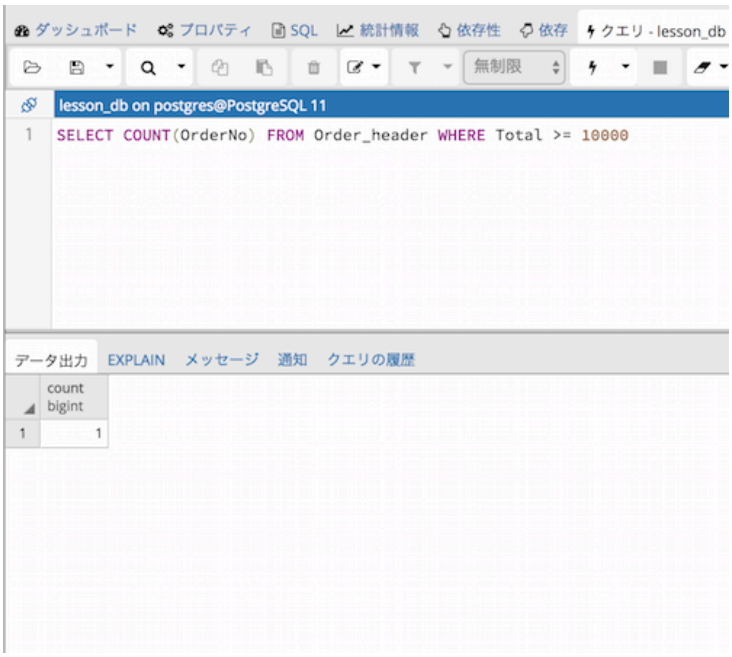
```
SELECT COUNT(OrderNo) FROM Order_header
```

?



では次に合計が10,000円以上の注文件数をカウントしてみましょう。

```
SELECT COUNT(OrderNo) FROM Order_header WHERE Total >= 10000
```



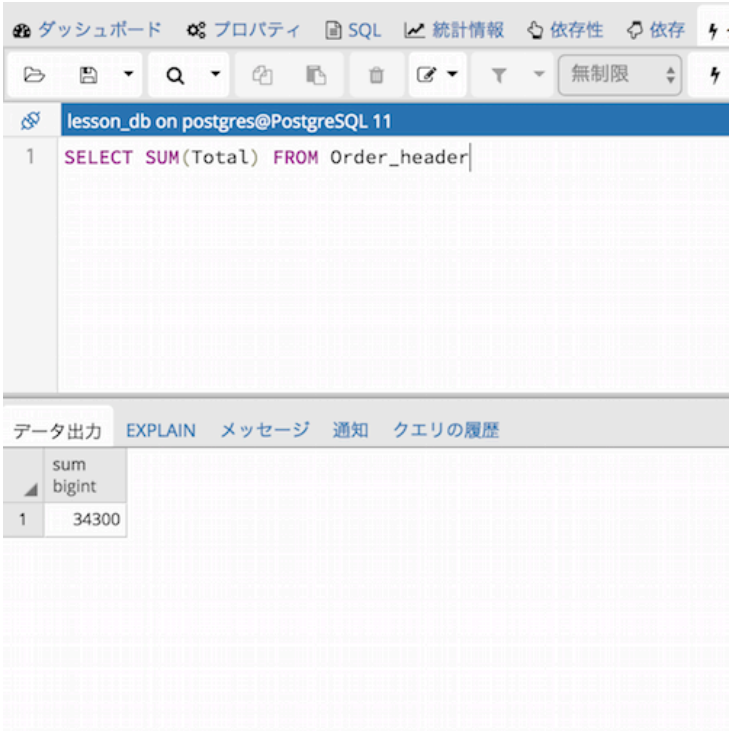
SUM()

SUM() は合計を出力します。

```
SELECT SUM(列名) FROM テーブル名
```

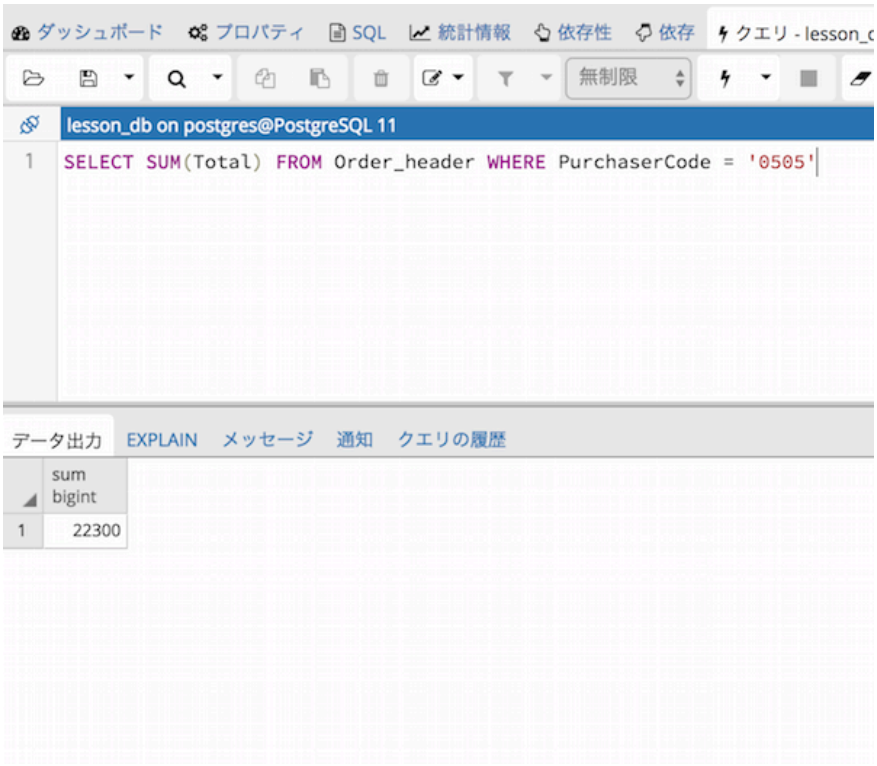
現在登録されている売上金額を全部足してみましょう。

```
SELECT SUM(Total) FROM Order_header
```



次は指定した購入者に対する、これまでの売上金額を合計します。

```
SELECT SUM(Total) FROM Order_header WHERE PurchaserCode = '0505'
```



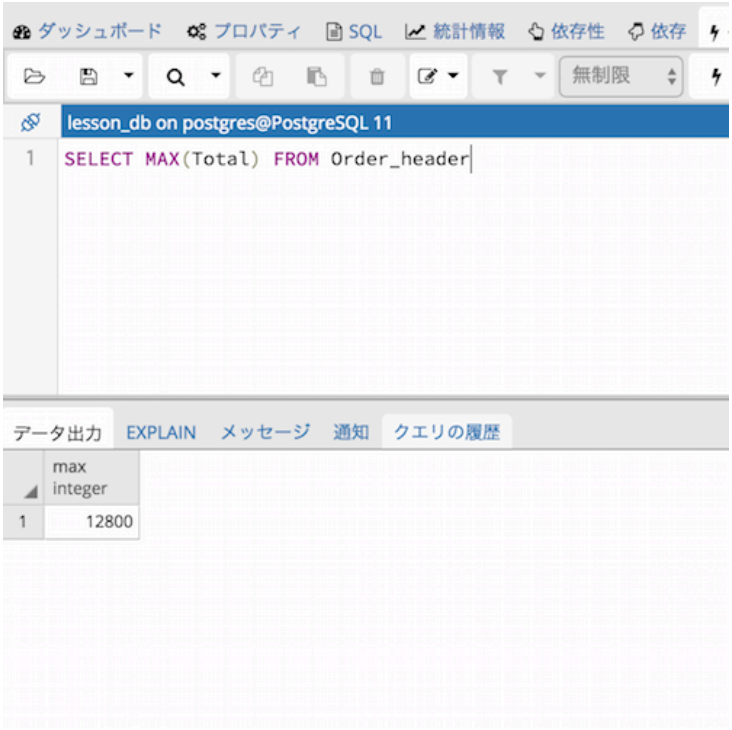
MAX()

MAX() は最大値を出力します。

```
SELECT MAX(列名) FROM テーブル名
```

注文のうち最も高い売上金額を表示します。

```
SELECT MAX(Total) FROM Order_header
```



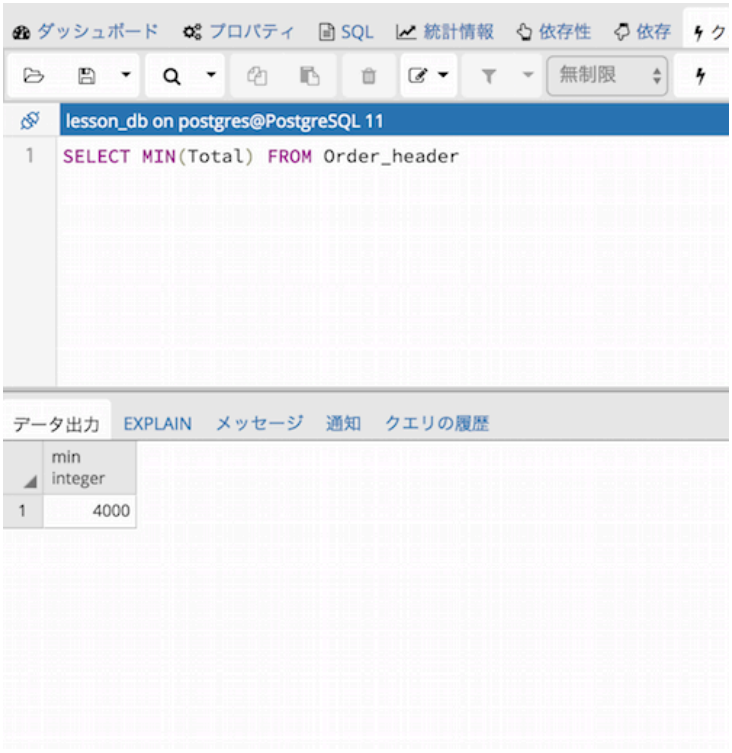
MIN()

MIN() は最小値を出力します。

```
SELECT MIN(列名) FROM テーブル名
```

注文のうち最も低い売上金額を表示します。

```
SELECT MIN(Total) FROM Order_header
```



AVG()

AVG() は平均を出力します。

```
SELECT AVG(列名) FROM テーブル名
```

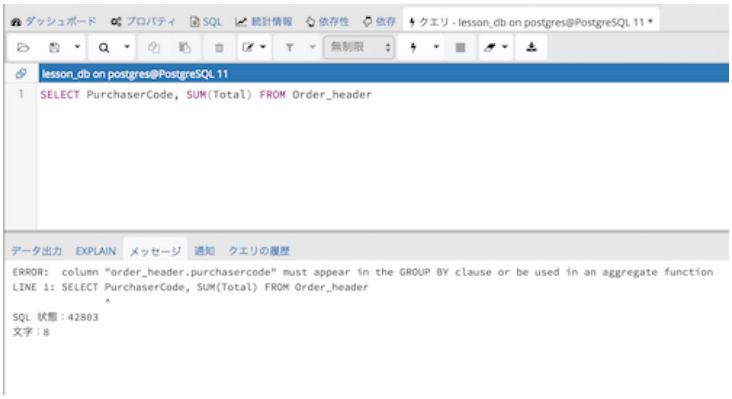
売上の平均金額を出力してみましょう。

```
SELECT AVG(Total) FROM Order_header
```

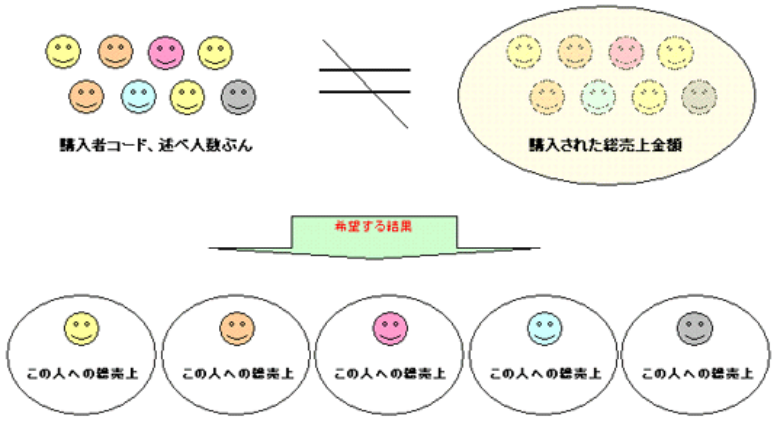
Step2 : GROUP BY

先ほどの例で、ある購入者に対する総売上金額を出力してみました。
もし数千人、数万人の購入者がいて一人一人SELECT文を実行するのは大変です。
各購入者の総売上金額の出力をひとつのSQL文で対処できないだろうか...
そんな時に役立つのが `GROUP BY`です。

```
SELECT PurchaserCode, SUM(Total) FROM Order_header
```



エラーです。それではSELECT文をよく見てみましょう。
SUM(Total)をSELECT文から消去すると、**「購入者コードを注文ヘッダーテーブルから抽出する」**という記述です。
それでは次にPurchaserCodeをSELECT文から消してみます。
「登録されている売上金額の合計を出力する」です。
ということは出力項目においてSELECT対象が明らかに違います。
知りたいのは購入者ごとの合計 です。

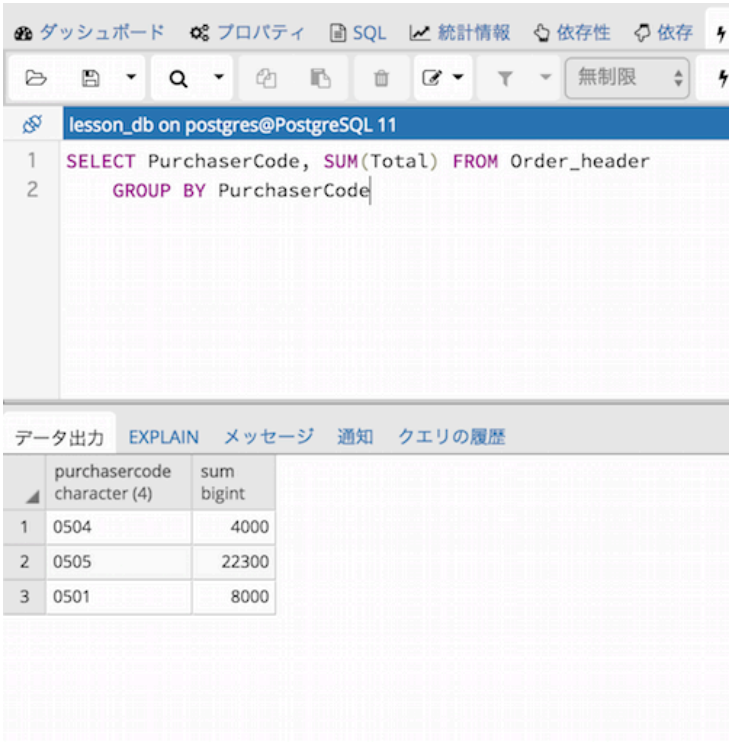


これを解決するのがGROUP BY句です。

```
SELECT 列名1, 集計関数(列名2) [, 列名3 … ] FROM テーブル名
    GROUP BY 列名1 [, 列名3 … ]
```

先ほどの例に当てはめてみましょう。

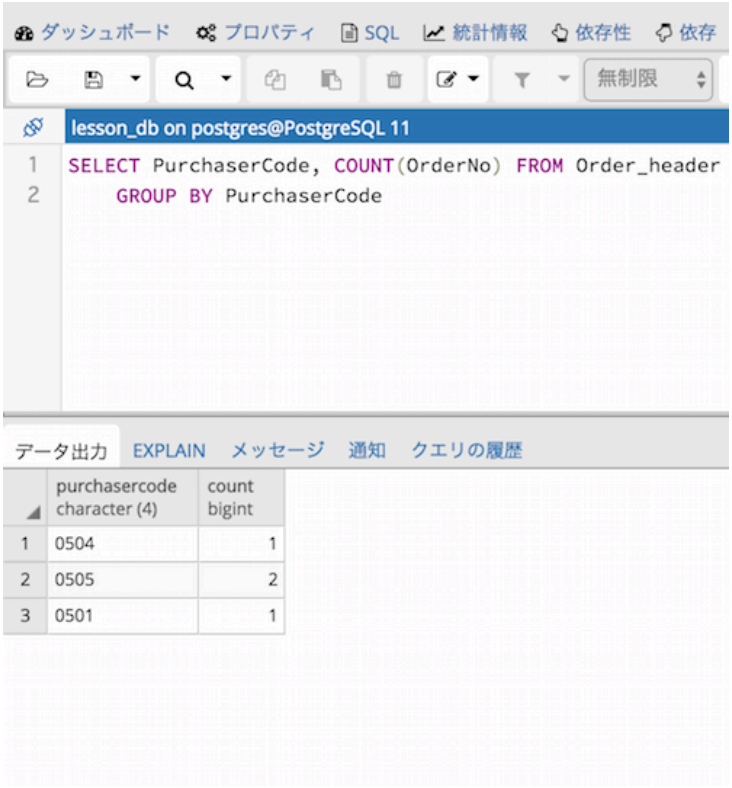
```
SELECT PurchaserCode, SUM(Total) FROM Order_header
    GROUP BY PurchaserCode
```



GROUP BY句にはまとめたい列名を記述します。この例では購入者ごとにまとめたいのでPurchaserCodeを指定します。
するとSUM(Total)の対象は全レコードではなく、PurchaserCodeでグループ分けされた
レコードとなります。同じくPurchaserCodeもPurchaserCodeでグループ分けされた列を
SELECTしているため、「購入者ごとの売上金額の合計」という結果が得られるのです。

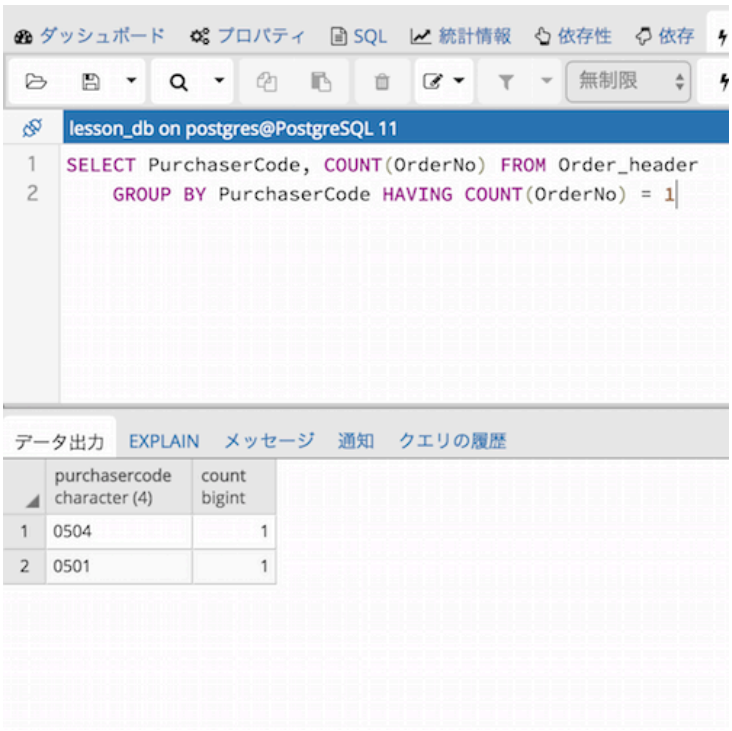
では同じくCOUNT()を使用して購入者ごとの注文回数を調べましょう。

```
SELECT PurchaserCode, COUNT(OrderNo) FROM Order_header
    GROUP BY PurchaserCode
```



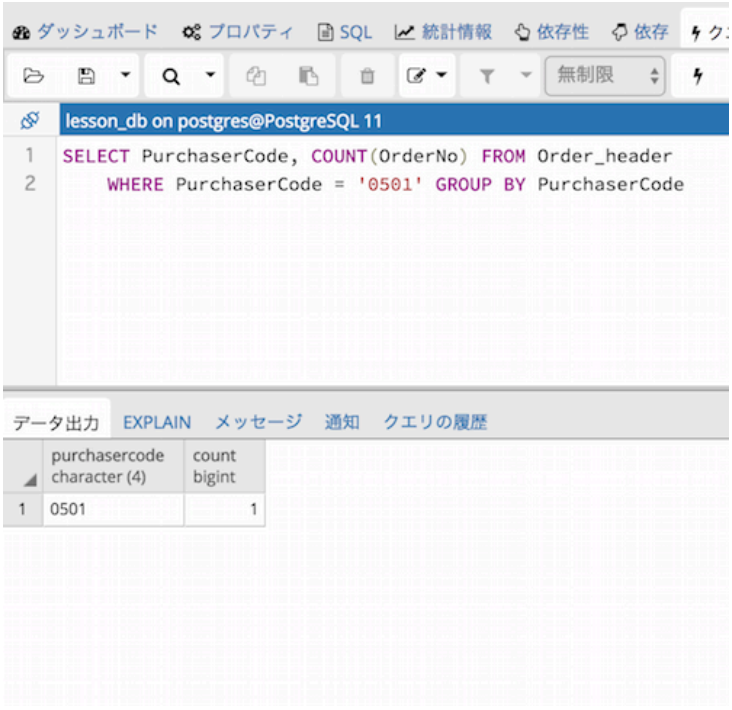
ここでもう少し条件を加えて、注文回数が1回の購入者を調べます。
集計された結果に対する条件はWHERE句では記述できません。 HAVING句を使います。

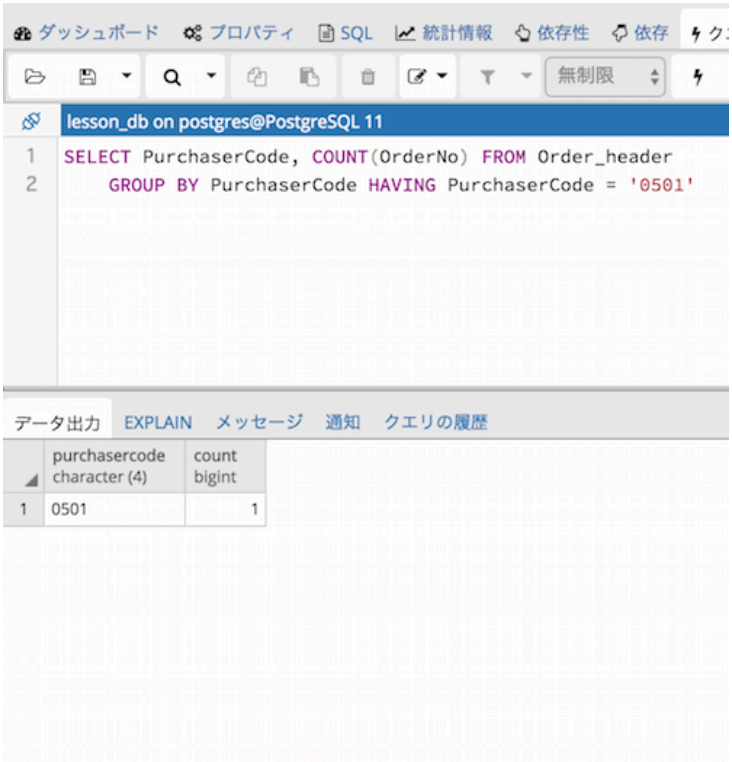
```
SELECT PurchaserCode, COUNT(OrderNo) FROM Order_header
    GROUP BY PurchaserCode HAVING COUNT(OrderNo) = 1
```



HAVING句の後ろに集計行に対する条件を指定します。集計関数が関わらない条件ならばWHERE句も使えます。
購入者を指定した購入件数を調べます。実行してみましょう。

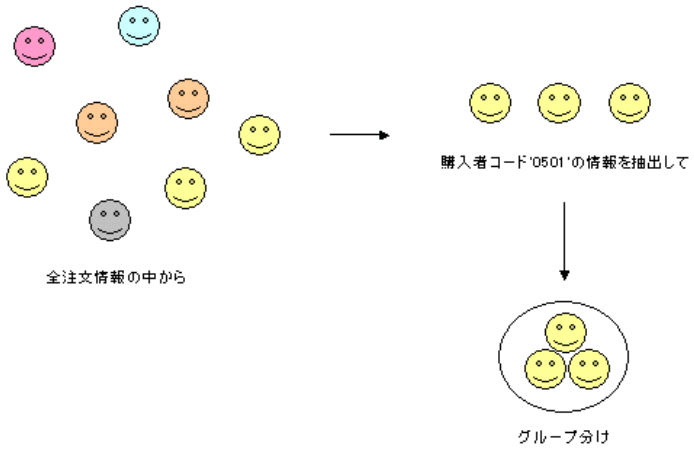
```
SELECT PurchaserCode, COUNT(OrderNo) FROM Order_header
    WHERE PurchaserCode = '0501' GROUP BY PurchaserCode

SELECT PurchaserCode, COUNT(OrderNo) FROM Order_header
    GROUP BY PurchaserCode HAVING PurchaserCode = '0501'
```



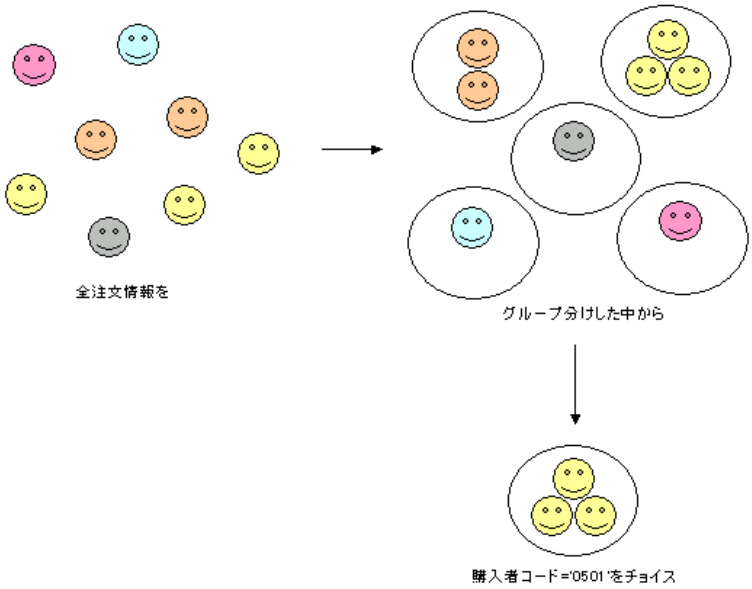


WHERE句でもHAVING句でも同じ結果になりました。結果は同じでも検索手順が異なります。
WHERE句のほうは購入者コード'0501'の情報を抽出後、購入者コードでグループ分けしています。
HAVING句では、まずグループ分けを行ってから購入者コード'0501'のグループを抽出します。

● WHERE句による検索手順



● HAVING句による検索手順



このように、集計関数がWHERE句で使用できないのは、グループ分けを行わない限り値を抽出できないからです。

副問い合わせ（サブクエリ）

ここまでテーブルを結合しての抽出、グループ分けしての抽出と色々な方法を実行してきました。
この章では、副問い合わせ（サブクエリ）を学んでいきます。
副問い合わせとは、 **「SELECT文の中でSELECT文を使う」** 方法です。
これを使えるようになると、いろいろな方法を組み合わせてSQLを実行できるので、
大抵のレコードは抽出できるようになります。

Step1 : サブクエリ

例えば、商品テーブルの商品のうち、平均単価より高い商品名を調べたい時はどうするのか...
副問い合わせを使用しない場合、最初のSELECT文で平均単価を調べたら、その値をどこかで覚えておいて
次のSELECT文で平均単価を条件として、求めるレコードを抽出しなければなりません。

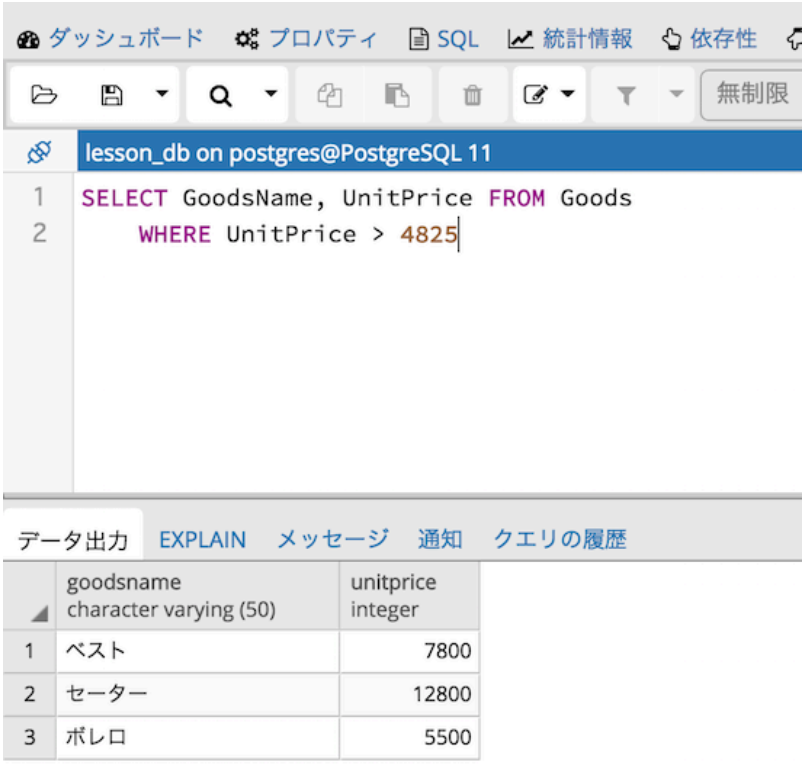
それぞれ記述してみましょう。

【１】商品の平均単価を調べる

```
SELECT AVG(UnitPrice) FROM Goods
```

【２】【１】の結果より単価が高いもの

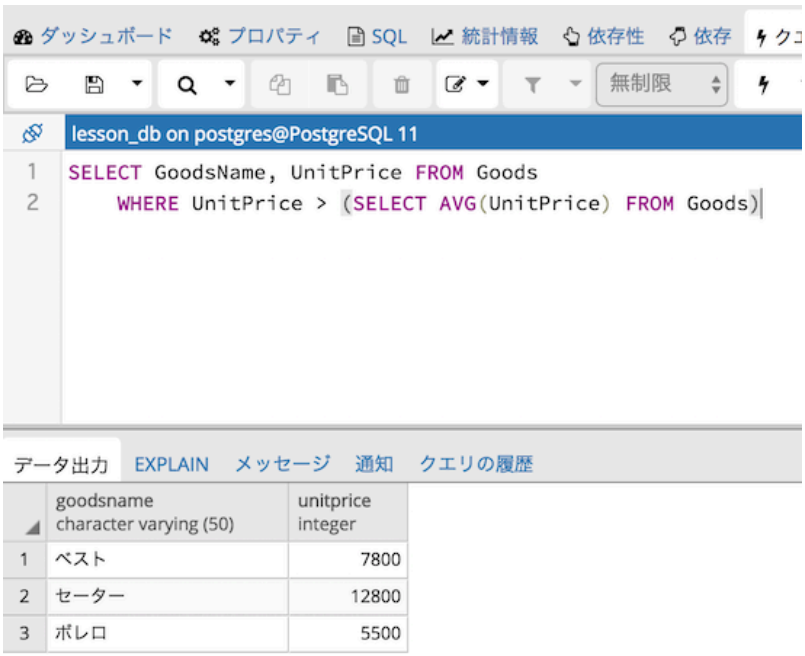
```
SELECT GoodsName, UnitPrice FROM Goods
    WHERE UnitPrice > 【１】の結果
```



	goodsname character varying (50)	unitprice integer
1	ベスト	7800
2	セーター	12800
3	ボレロ	5500

↓ 副問い合わせを使うと一度で実行できます。

```
SELECT GoodsName, UnitPrice FROM Goods
    WHERE UnitPrice > (SELECT AVG(UnitPrice) FROM Goods)
```



	goodsname character varying (50)	unitprice integer
1	ベスト	7800
2	セーター	12800
3	ボレロ	5500

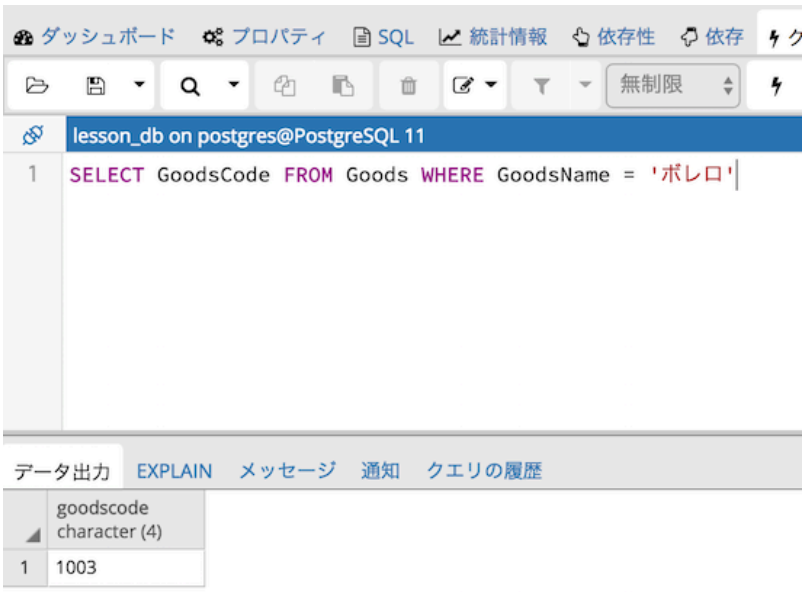
解説

項番【2】の「【1】の結果」の部分に項番【1】のSELECT文を括弧でくくって記述しています。
括弧でくくったSELECT文がサブクエリです。 サブクエリに対しておおもとのSELECT文をメインクエリと呼びます。

次は、ある商品を購入した際の注文番号を調べます。商品名から注文番号を知りたいのですが、
注文詳細テーブルには商品名情報がありません。商品コードは存在するので、まず最初に商品テーブルで
商品名から商品コードを調べる必要があります。

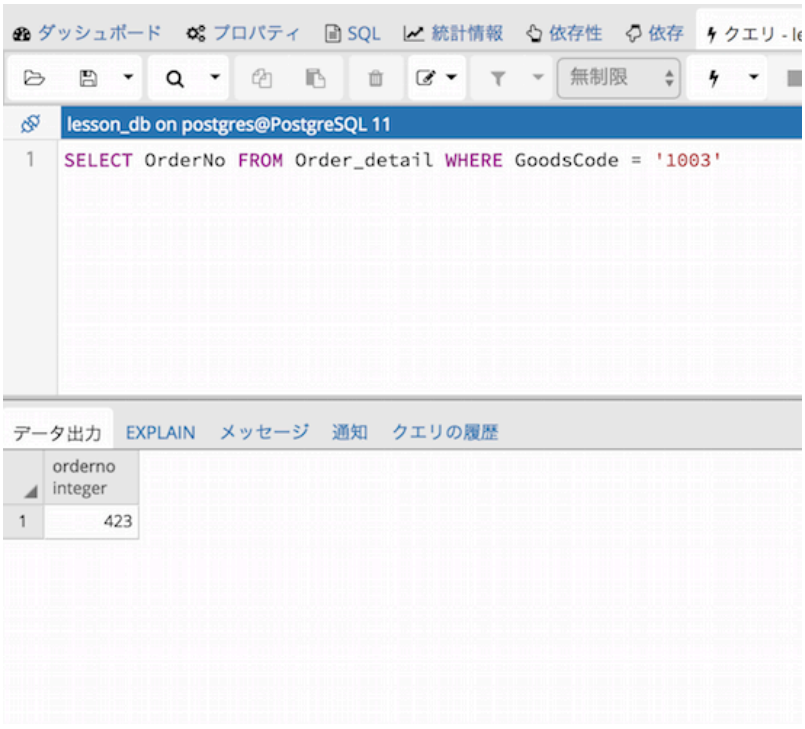
【１】商品名から商品コードを調べる

```
SELECT GoodsCode FROM Goods WHERE GoodsName = 'ボレロ'
```



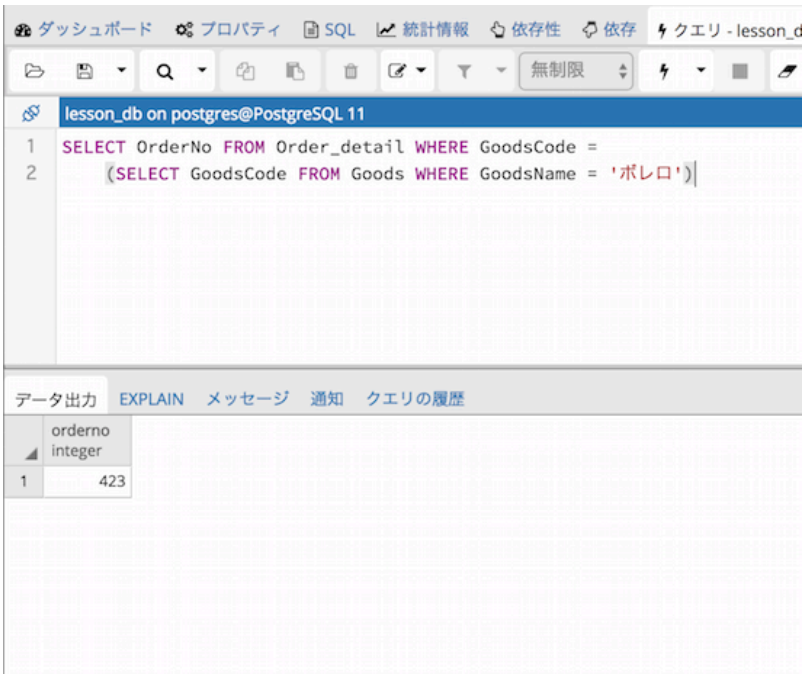
【２】【１】の結果から注文詳細テーブルで注文番号を調べる

```
SELECT OrderNo FROM Order_detail WHERE GoodsCode = 【１】の結果
```



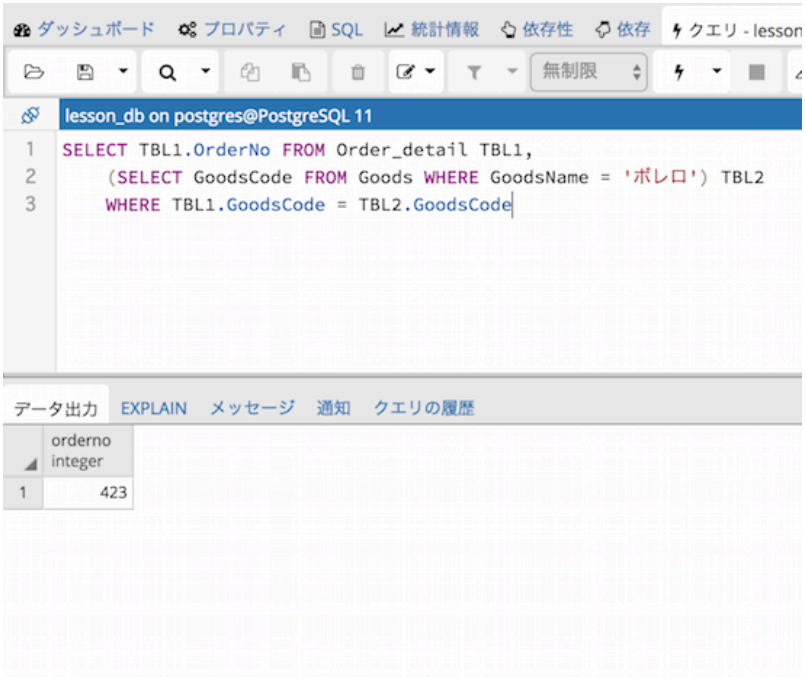
↓　副問い合わせにしてみる。

```
SELECT OrderNo FROM Order_detail WHERE GoodsCode =
    (SELECT GoodsCode FROM Goods WHERE GoodsName = 'ボレロ')
```



最初の例題と同じパターンです。使い方がわかってきましたか？
少々パターンを変えてみましょう。次のSQL文は上と同じ結果を返します。

```
SELECT TBL1.OrderNo FROM Order_detail TBL1,
    (SELECT GoodsCode FROM Goods WHERE GoodsName = 'ボレロ') TBL2
    WHERE TBL1.GoodsCode = TBL2.GoodsCode
```

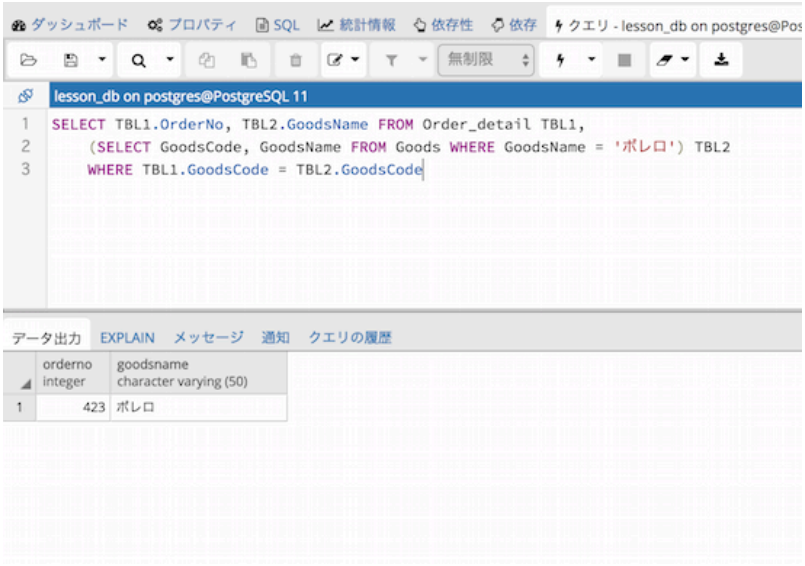


解説

先ほどはWHERE句にサブクエリを記述しましたが、今度はFROM句に書きました。
副問い合わせの結果を一時的にテーブルのように保持 し、
FROM句で指定した他のテーブルと等価結合して結果を引き出します。

FROM句にサブクエリを記述した場合、テーブル名はサブクエリ内のGoodsではなくなるため別名として
TBL2を指定しています。Order_detailはそのままでも構いません。
ここではわかりやすいようにTBL1という別名にしました。

```
SELECT TBL1.OrderNo, TBL2.GoodsName FROM Order_detail TBL1,
    (SELECT GoodsCode, GoodsName FROM Goods WHERE GoodsName = 'ボレロ') TBL2
    WHERE TBL1.GoodsCode = TBL2.GoodsCode
```

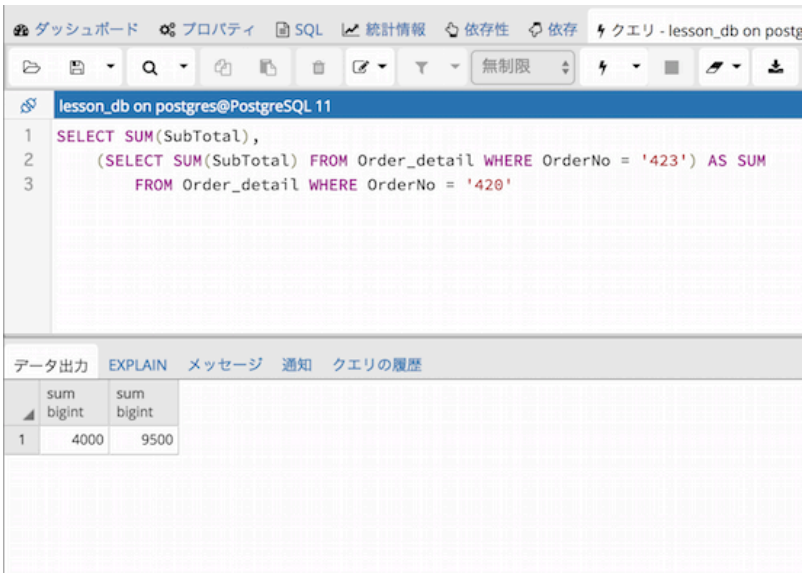


解説

FROM句の場合、一時テーブルを作成するイメージなので、
メインクエリのSELECTにサブクエリの結果を出力することができます。

またSELECT句にもサブクエリを使うことが出来ます。

```
SELECT SUM(SubTotal),
    (SELECT SUM(SubTotal) FROM Order_detail WHERE OrderNo = '423') AS SUM
        FROM Order_detail WHERE OrderNo = '420'
```



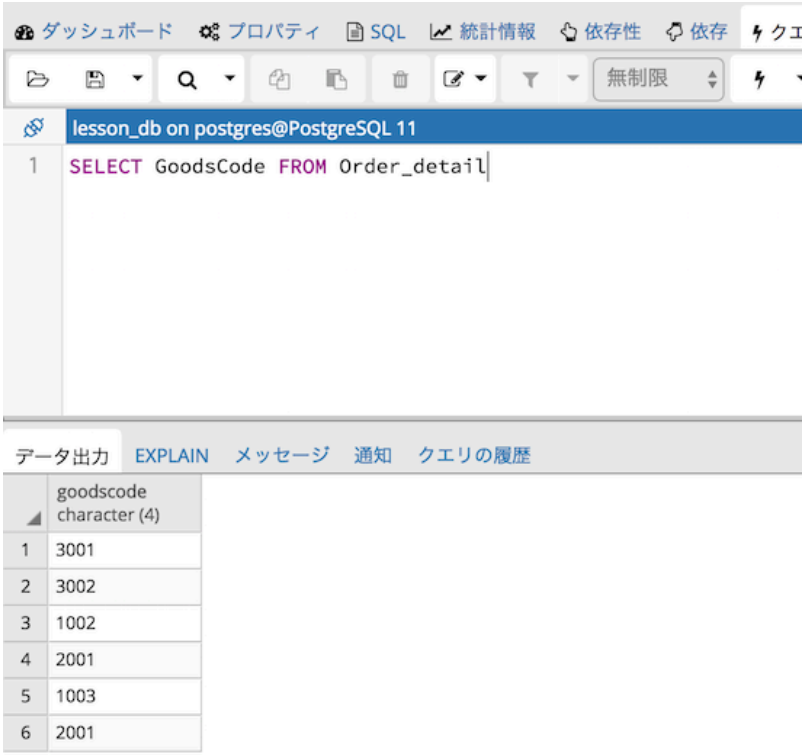
解説

注文番号420、注文番号423の小計の合計を出力しています。
副問い合わせで抽出した列には名前が付いていないので、SUMと名前を付けています。

では、次は注文詳細テーブルに登録のある商品の全情報を商品テーブルから出力してみましょう。

【１】注文詳細テーブルの商品コードを調べる

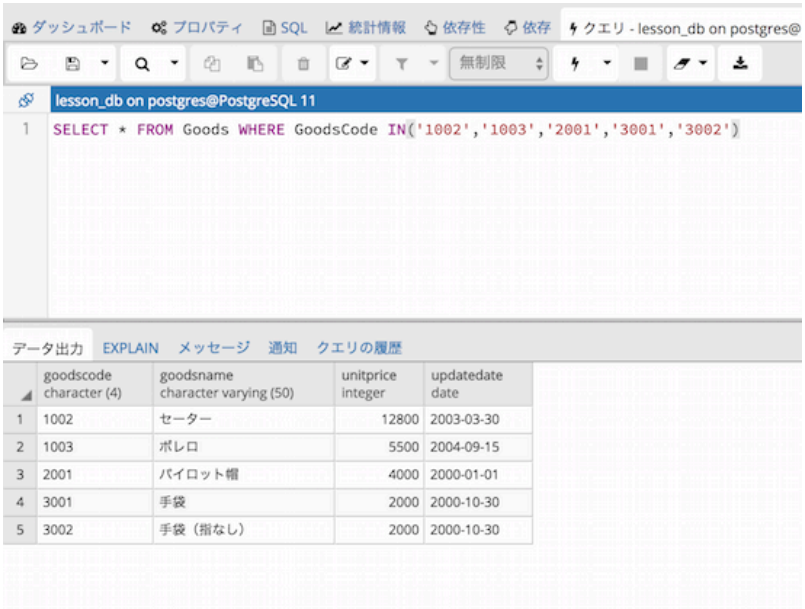
```
SELECT GoodsCode FROM Order_detail
```



	goodscode character (4)
1	3001
2	3002
3	1002
4	2001
5	1003
6	2001

【２】【１】の結果から商品情報を調べる

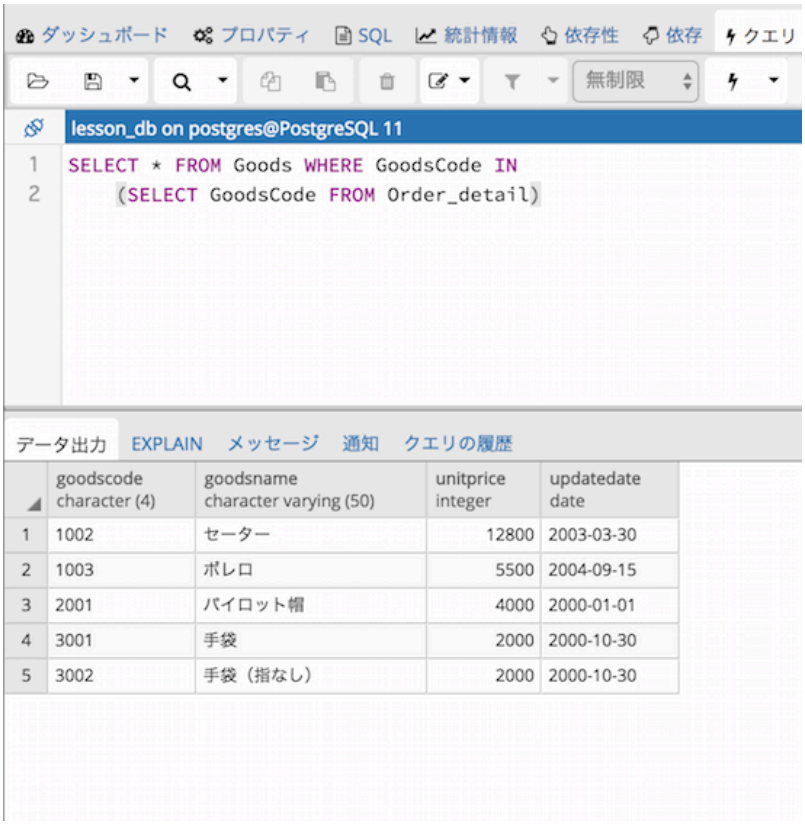
```
SELECT * FROM Goods WHERE GoodsCode IN(【１】の結果)
```



	goodscode character (4)	goodsname character varying (50)	unitprice integer	updatedate date
1	1002	セーター	12800	2003-03-30
2	1003	ボレロ	5500	2004-09-15
3	2001	パイロット帽	4000	2000-01-01
4	3001	手袋	2000	2000-10-30
5	3002	手袋（指なし）	2000	2000-10-30

↓　副問い合わせにしてみる。

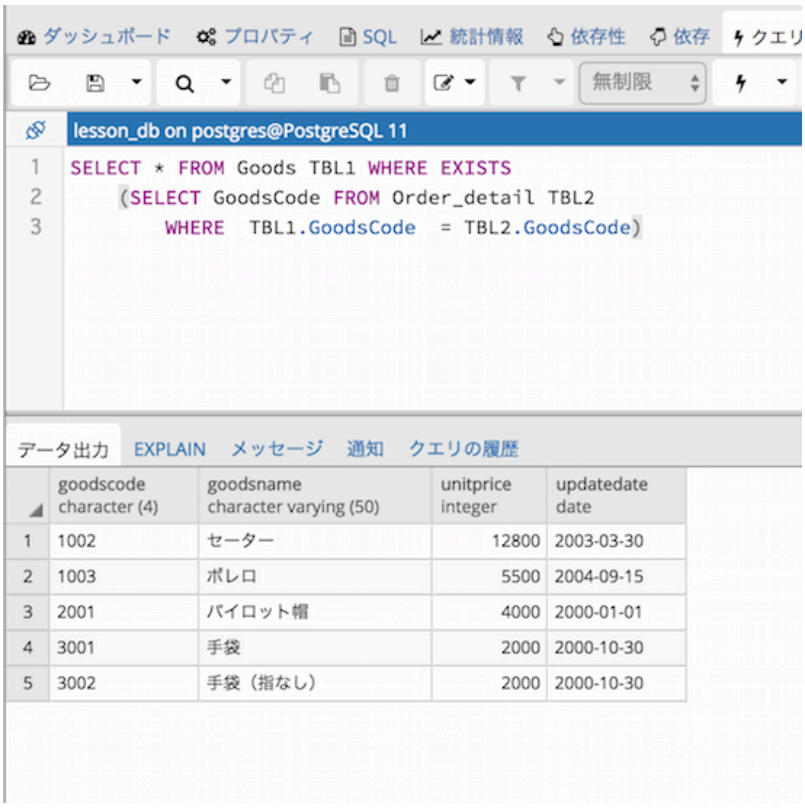
```
SELECT * FROM Goods WHERE GoodsCode IN
    (SELECT GoodsCode FROM Order_detail)
```



	goodscode character (4)	goodsname character varying (50)	unitprice integer	updatedate date
1	1002	セーター	12800	2003-03-30
2	1003	ボレロ	5500	2004-09-15
3	2001	パイロット帽	4000	2000-01-01
4	3001	手袋	2000	2000-10-30
5	3002	手袋（指なし）	2000	2000-10-30

この場合、商品テーブルで条件となるのは注文詳細テーブルに登録があるかないかです。
存在有無を判定するためのEXISTSという記述も使うことができます。

```
SELECT * FROM Goods TBL1 WHERE EXISTS
    (SELECT GoodsCode FROM Order_detail TBL2
        WHERE  TBL1.GoodsCode  = TBL2.GoodsCode)
```



	goodscode character (4)	goodsname character varying (50)	unitprice integer	updatedate date
1	1002	セーター	12800	2003-03-30
2	1003	ボレロ	5500	2004-09-15
3	2001	パイロット帽	4000	2000-01-01
4	3001	手袋	2000	2000-10-30
5	3002	手袋（指なし）	2000	2000-10-30

解説

・IN句の場合

1: サブクエリで抽出した内容がIN句へ
2: メインクエリのSELECT文の条件となり結果を抽出する
という流れとなります。

・EXISTSの場合

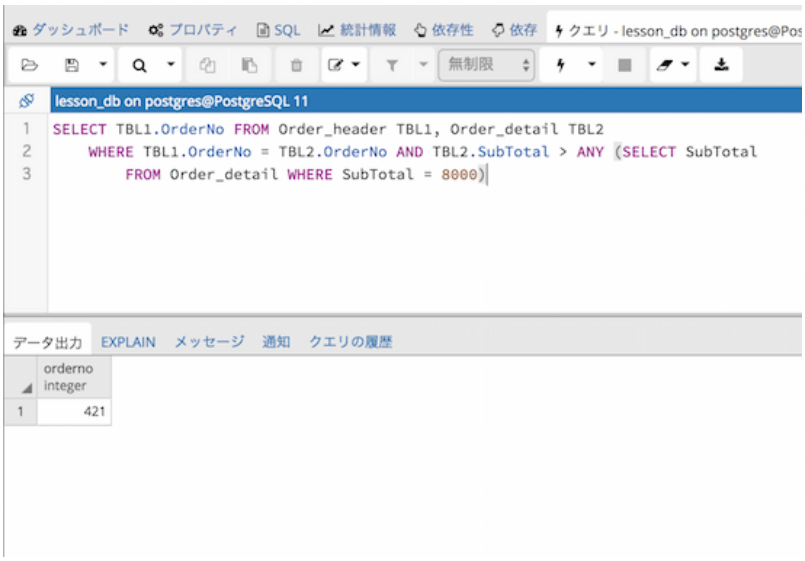
1: メインクエリのテーブルレコードを一行ずつサブクエリへ渡す
2: サブクエリ内のテーブルレコードと等価条件による比較を行う
3: 一致したかどうかを `TRUE`、`FALSE` でメインクエリに返し、`TRUE` ならば出力する
という流れとなります。

SQL文を見ると、サブクエリ内にメインクエリのテーブルである `TBL1` という記述がありますね。
このように「サブクエリでメインクエリを参照するもの」を「 **相関サブクエリ** 」と呼びます。

Step 2 : ANY句

サブクエリに不等号を使用することが出来ます。

```
SELECT TBL1.OrderNo FROM Order_header TBL1, Order_detail TBL2
    WHERE TBL1.OrderNo = TBL2.OrderNo AND TBL2.SubTotal > ANY (SELECT SubTotal
        FROM Order_detail WHERE SubTotal = 8000)
```



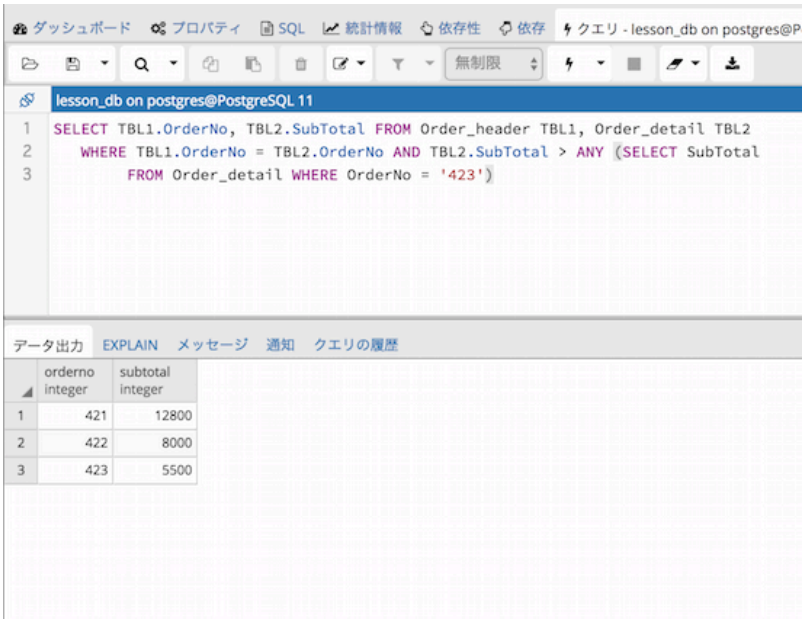
> **解説**

ANY句ではIN句では使えなかった、不等号が使えます。

1. サブクエリで小計が8000の小計を抽出
2. ANY句を使い、抽出した小計より大きい小計の注文番号を取得

ANY句では **サブクエリで生成した値のいずれかの値を評価対象** としています。

```
SELECT TBL1.OrderNo, TBL2.SubTotal FROM Order_header TBL1, Order_detail TBL2
   WHERE TBL1.OrderNo = TBL2.OrderNo AND TBL2.SubTotal > ANY (SELECT SubTotal
        FROM Order_detail WHERE OrderNo = '423')
```



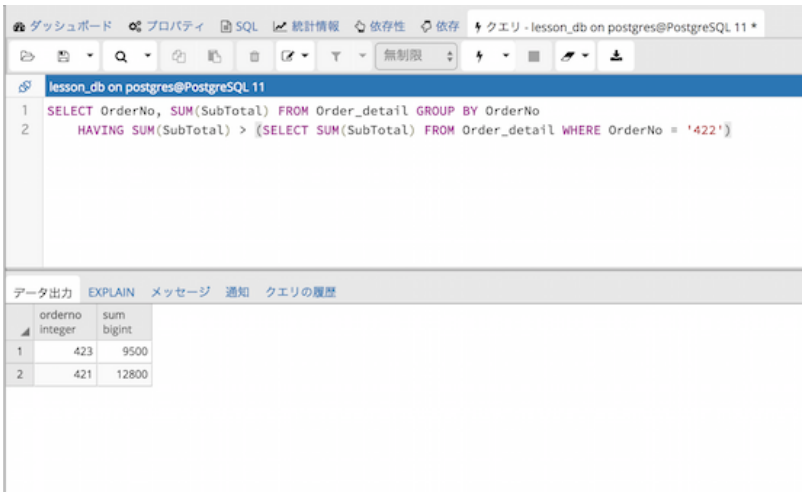
解説

1. サブクエリで注文番号が423の小計を抽出
2. ANY句を使い抽出した小計より大きい小計の注文番号を出力

サブクエリで取得した値は｛4000. 5500｝です。
この値のうちいずれかについて、注文詳細テーブルの各行の受注個数が大きくなる行を出力します。
今回の場合は4000より大きい値を出力しています。

HAVING句でもサブクエリを使用することが出来ます。

```
SELECT OrderNo, SUM(SubTotal) FROM Order_detail GROUP BY OrderNo
    HAVING SUM(SubTotal) > (SELECT SUM(SubTotal) FROM Order_detail WHERE OrderNo = '422')
```



注文番号ごとに小計の合計を出力します。サブクエリで注文番号が422の小計の合計を抽出しています。
サブクエリで抽出した小計の合計より大きい値を持つ、注文番号と小計の合計を出力しています。

クエリの結合

クエリの結合とは、2つのクエリの出力結果を「和、差、積、の 3 種類の方法で結合すること」をいいます。
サブクエリに比べると考え方は単純ですが、注意しなければいけない点があります。
それは、それぞれのクエリから出力される列は比較可能なものでなければなりません。

要するに、 **データ型** と **列数** が一致しなければ、クエリを結合することはできません。
その点を踏まえて、この章を進めて行きましょう！

Step1: 演算子の種類

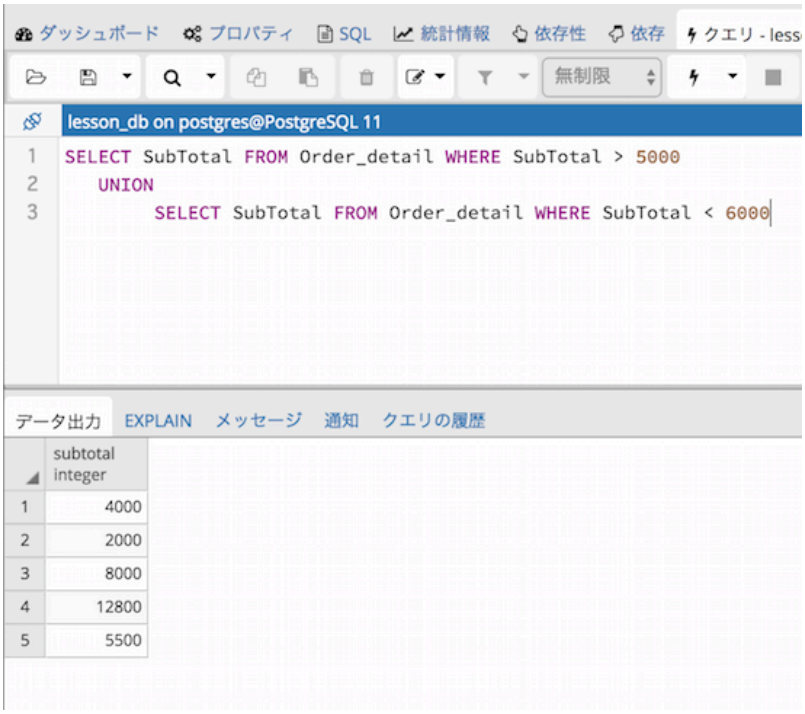
演算子の種類と内容を表にしました。

演算子	内容
UNION演算子	和結合
INTERSECT演算子	積結合
EXCEPT演算子	差結合

Step2 : UNION演算子

UNION演算子は、2つのクエリの出力結果の和集合を行います。
2つのクエリの出力結果を結合し、重複している行を削除します。

```
SELECT SubTotal FROM Order_detail WHERE SubTotal > 5000
   UNION
        SELECT SubTotal FROM Order_detail WHERE SubTotal < 6000
```

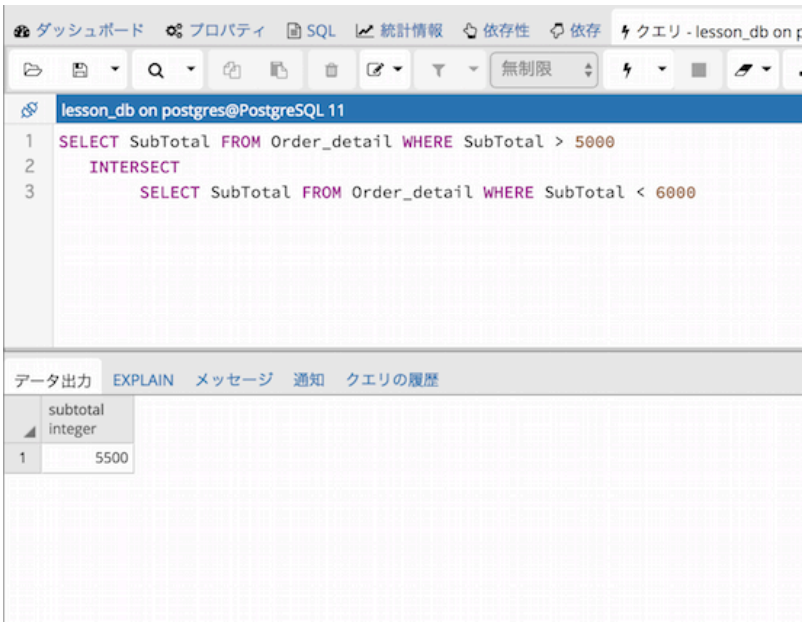


小計が5000より大きい値を出力するクエリと、小計が6000より小さい値を出力するクエリの和結合を出力します。

Step3 : INTERSECT演算子

INTERSECT演算子は、2つのクエリの出力結果の積集合を行います。
2つのクエリの出力結果から同じものだけを出力します。

```
SELECT SubTotal FROM Order_detail WHERE SubTotal > 5000
   INTERSECT
        SELECT SubTotal FROM Order_detail WHERE SubTotal < 6000
```

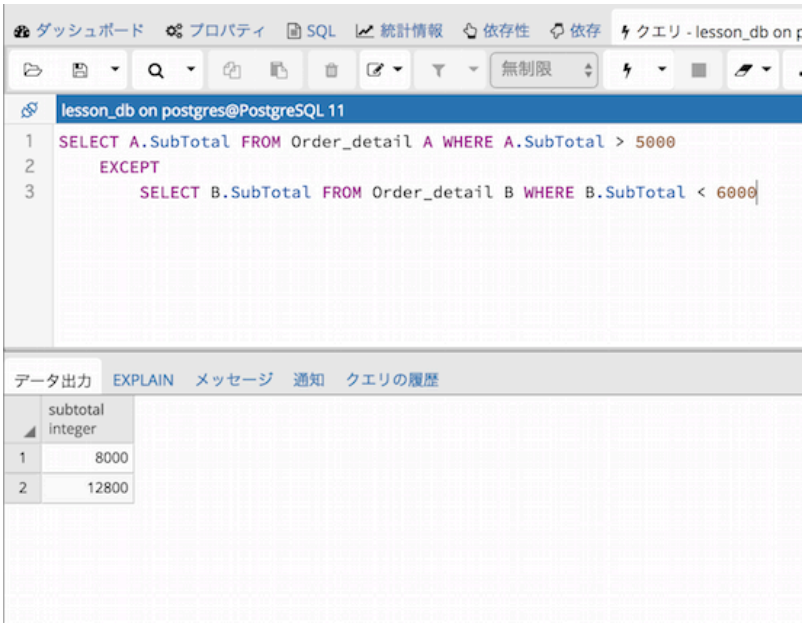


2つのクエリの出力結果の中で、同じものだけを出力します。

Step4 : EXCEPT演算子

EXCEPT演算子は、2つのクエリの出力結果の差集合を行います。

```
SELECT A.SubTotal FROM Order_detail A WHERE A.SubTotal > 5000
    EXCEPT
        SELECT B.SubTotal FROM Order_detail B WHERE B.SubTotal < 6000
```



テーブルＡの出力結果の内、テーブルＢの出力結果に含まれないものを出力します。
テーブルＡの出力結果は小計｛5500,8000,12800｝です。この中で小計｛5500｝はテーブルＢの出力結果に含まれているので出力しませんでした。

課題

提出課題はありませんので、一通り学習が終わったら次の章に進んで下さい。

最終更新日時: 2024年 05月 27日(月曜日) 14:53