

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
```

The screenshot shows a database management tool interface. At the top, there are tabs for 'ダッシュボード', 'プロパティ', 'SQL', '統計情報', '依存性', '依存', and 'クエリ'. Below these is a toolbar with icons for file operations, search, and execution. The main area displays a SQL query: `SELECT COUNT(OrderNo) FROM Order_header`. Below the query, there are tabs for 'データ出力', 'EXPLAIN', 'メッセージ', '通知', and 'クエリの履歴'. The 'データ出力' tab is active, showing a table with one row and one column. The column is labeled 'count' and 'bigint'. The row contains the value '4'.

	count bigint
1	4

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
```

The screenshot shows the same database management tool interface. The SQL query is now: `SELECT COUNT(OrderNo) FROM Order_header WHERE Total >= 10000`. The 'データ出力' tab is active, showing a table with one row and one column. The column is labeled 'count' and 'bigint'. The row contains the value '1'.

	count bigint
1	1

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
```

The screenshot shows a database management tool interface. At the top, there are tabs for 'ダッシュボード', 'プロパティ', 'SQL', '統計情報', '依存性', and '依存'. Below these is a toolbar with icons for file operations and a search bar. The main area displays a SQL query: `SELECT SUM(Total) FROM Order_header`. Below the query, there are tabs for 'データ出力', 'EXPLAIN', 'メッセージ', '通知', and 'クエリの履歴'. The 'データ出力' tab is selected, showing a table with one row:

	sum bigint
1	34300

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'
```

The screenshot shows the same database management tool interface. The SQL query is now: `SELECT SUM(Total) FROM Order_header WHERE PurchaserCode = '0505'`. The 'データ出力' tab shows a table with one row:

	sum bigint
1	22300

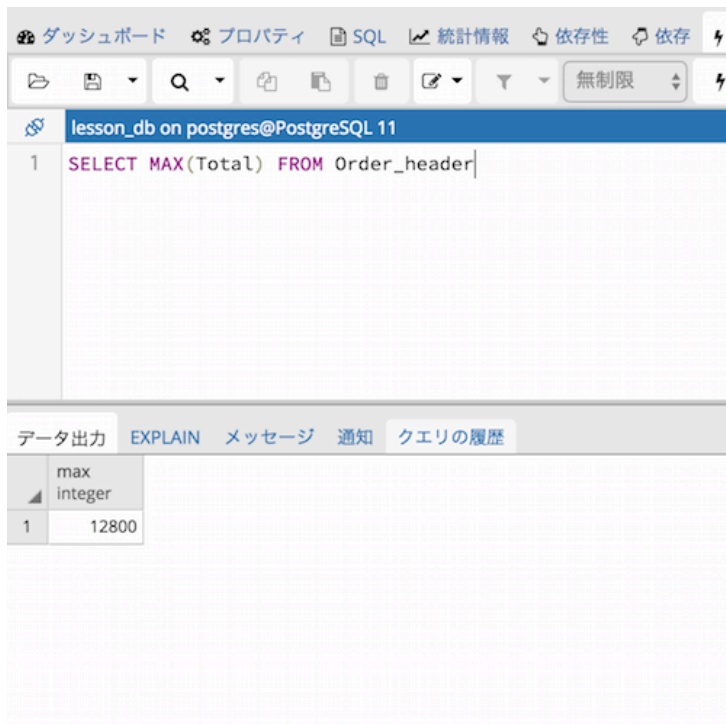
MAX()

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
```



The screenshot shows a database management tool interface. At the top, there's a navigation bar with icons for Dashboard, Properties, SQL, Statistics, Dependencies, and Favorites. Below this is a toolbar with icons for file operations and a search bar. The main area displays a SQL query: `SELECT MAX(Total) FROM Order_header`. Below the query, there's a tabbed interface with 'データ出力' (Data Output) selected. The output shows a single row with the value 12800. The data type is 'max integer'.

	max integer
1	12800

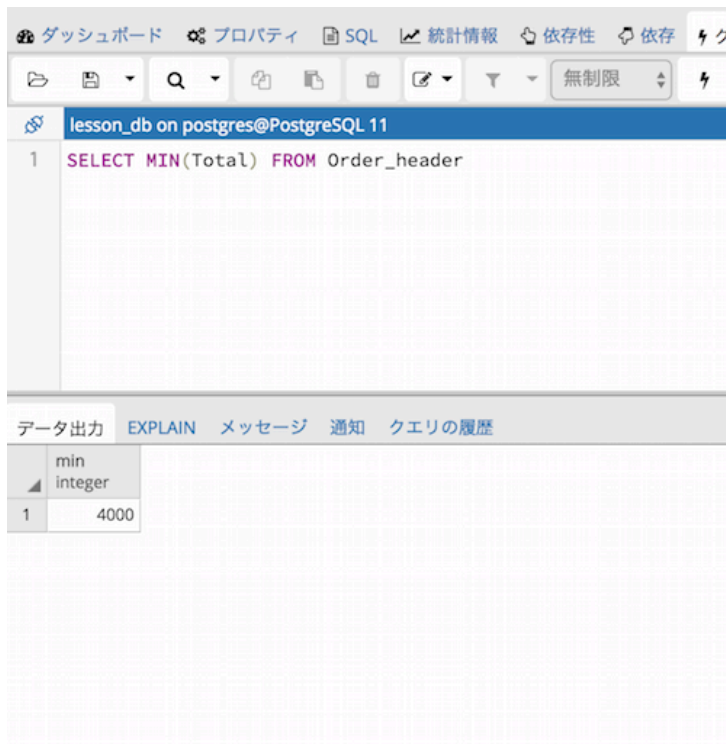
MIN()

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
```



The screenshot shows the same database management tool interface. The SQL query is now: `SELECT MIN(Total) FROM Order_header`. The output shows a single row with the value 4000. The data type is 'min integer'.

	min integer
1	4000

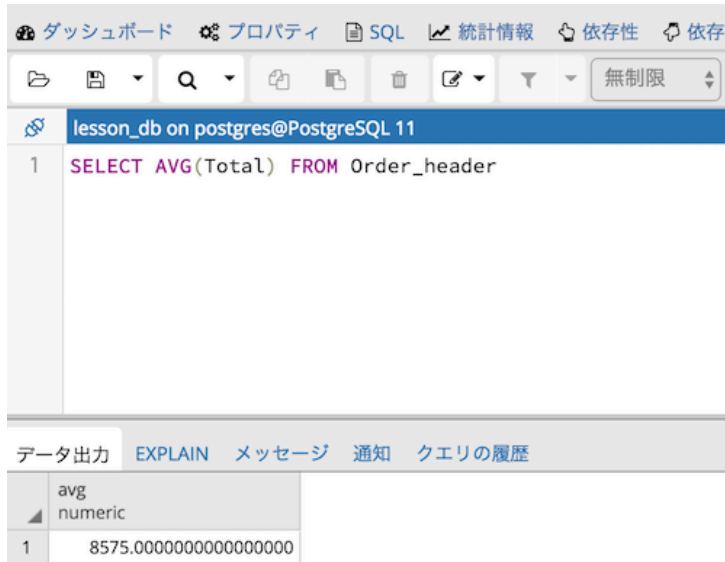
AVG()

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
```



ダッシュボード プロパティ SQL 統計情報 依存性 依存

lesson_db on postgres@PostgreSQL 11

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

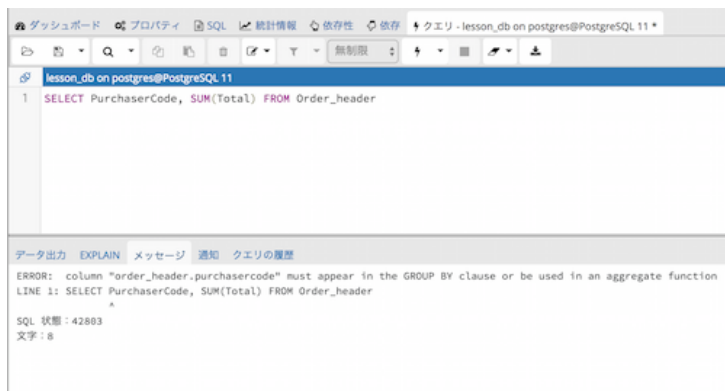
データ出力 EXPLAIN メッセージ 通知 クエリの履歴

	avg
1	8575.0000000000000000

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
```



ダッシュボード プロパティ SQL 統計情報 依存性 依存 クエリ - lesson_db on postgres@PostgreSQL 11 *

lesson_db on postgres@PostgreSQL 11

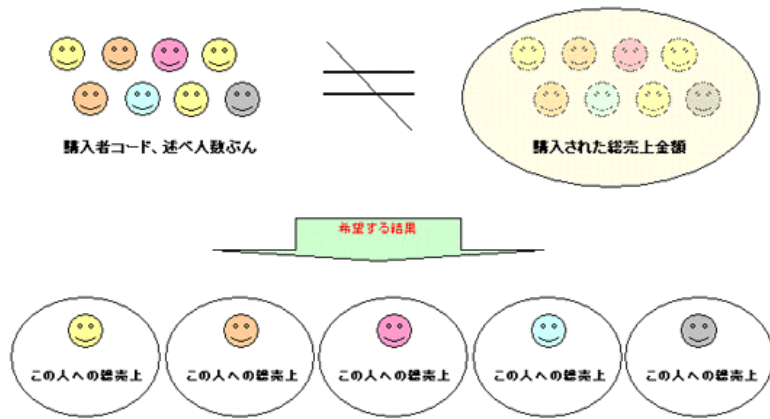
```
1 SELECT PurchaserCode, SUM(Total) FROM Order_header
```

データ出力 EXPLAIN メッセージ 通知 クエリの履歴

ERROR: column "order_header.purchasercode" must appear in the GROUP BY clause or be used in an aggregate function
LINE 1: SELECT PurchaserCode, SUM(Total) FROM Order_header

SQL 状態: 42803
文字: 8

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
```

	purchasercode character (4)	sum bigint
1	0504	4000
2	0505	22300
3	0501	8000

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
```

ダッシュボード プロパティ SQL 統計情報 依存性 依存

lesson_db on postgres@PostgreSQL 11

```

1 SELECT PurchaserCode, COUNT(OrderNo) FROM Order_header
2   GROUP BY PurchaserCode

```

データ出力 EXPLAIN メッセージ 通知 クエリの履歴

	purchasercode character (4)	count bigint
1	0504	1
2	0505	2
3	0501	1

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

```

ダッシュボード プロパティ SQL 統計情報 依存性 依存

lesson_db on postgres@PostgreSQL 11

```

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

```

データ出力 EXPLAIN メッセージ 通知 クエリの履歴

	purchasercode character (4)	count bigint
1	0504	1
2	0501	1

HAVING句の後ろに集計行に対する条件を指定します。集計関数に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'

```


The screenshot shows a PostgreSQL query editor interface. The top navigation bar includes tabs for 'ダッシュボード' (Dashboard), 'プロパティ' (Properties), 'SQL', '統計情報' (Statistics), '依存性' (Dependencies), and '依存' (Dependencies). Below the navigation bar is a toolbar with icons for file operations and a search bar. The main editor area displays a SQL query:

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

Below the query editor, there are tabs for 'データ出力' (Data Output), 'EXPLAIN', 'メッセージ' (Messages), '通知' (Notifications), and 'クエリの履歴' (Query History). The 'データ出力' tab is active, showing a table with the following data:

	purchasercode character (4)	count bigint
1	0501	1

The screenshot shows the same PostgreSQL query editor interface. The main editor area displays a SQL query:

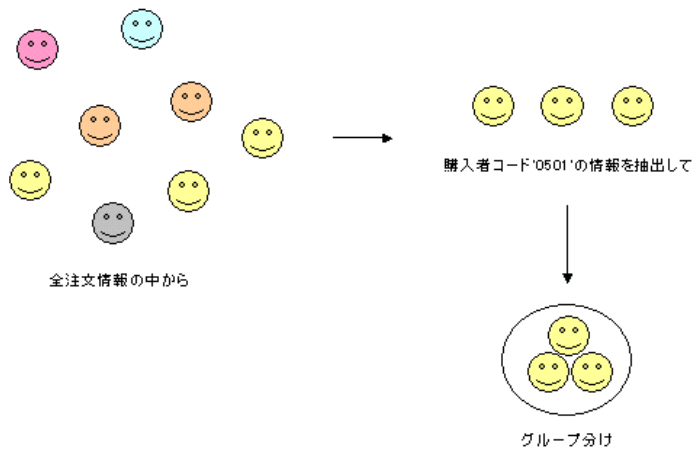
```
1 SELECT PurchaserCode, COUNT(OrderNo) FROM Order_header
2 GROUP BY PurchaserCode HAVING PurchaserCode = '0501'
```

Below the query editor, the 'データ出力' (Data Output) tab is active, showing a table with the following data:

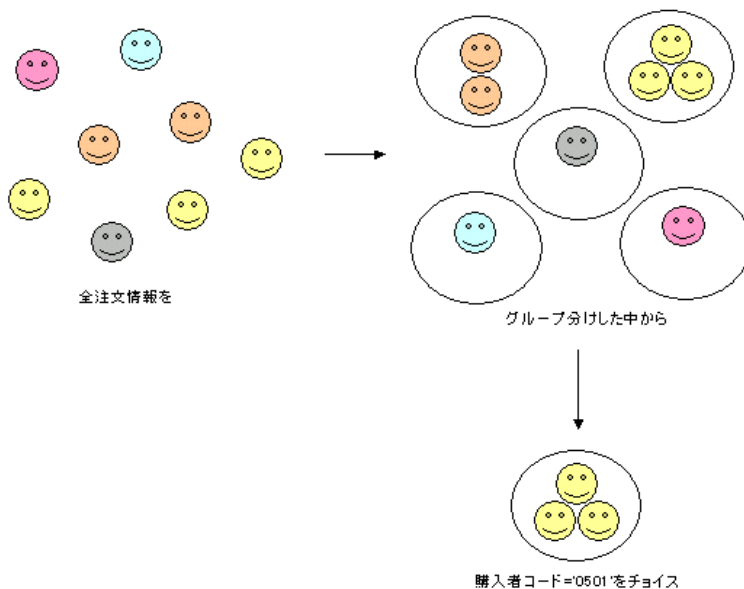
	purchasercode character (4)	count bigint
1	0501	1

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
```

The screenshot shows a web-based SQL editor for PostgreSQL 11. The interface includes a top navigation bar with icons for Dashboard, Properties, SQL, Statistics, and Dependencies. Below this is a toolbar with icons for file operations and a '無制限' (Unlimited) button. The main editor area shows a query: `SELECT AVG(UnitPrice) FROM Goods`. Below the editor, there are tabs for 'データ出力' (Data Output), 'EXPLAIN', 'メッセージ' (Messages), '通知' (Notifications), and 'クエリの履歴' (Query History). The 'データ出力' tab is active, displaying a table with the query result.

	avg numeric
1	5300.0000000000000000

【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
```

ダッシュボード プロパティ SQL 統計情報 依存性

lesson_db on postgres@PostgreSQL 11

```
1 SELECT GoodsName, UnitPrice FROM Goods
2 WHERE UnitPrice > 4825
```

データ出力 EXPLAIN メッセージ 通知 クエリの履歴

	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)
```

ダッシュボード プロパティ SQL 統計情報 依存性 依存 クコ

lesson_db on postgres@PostgreSQL 11

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

データ出力 EXPLAIN メッセージ 通知 クエリの履歴

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

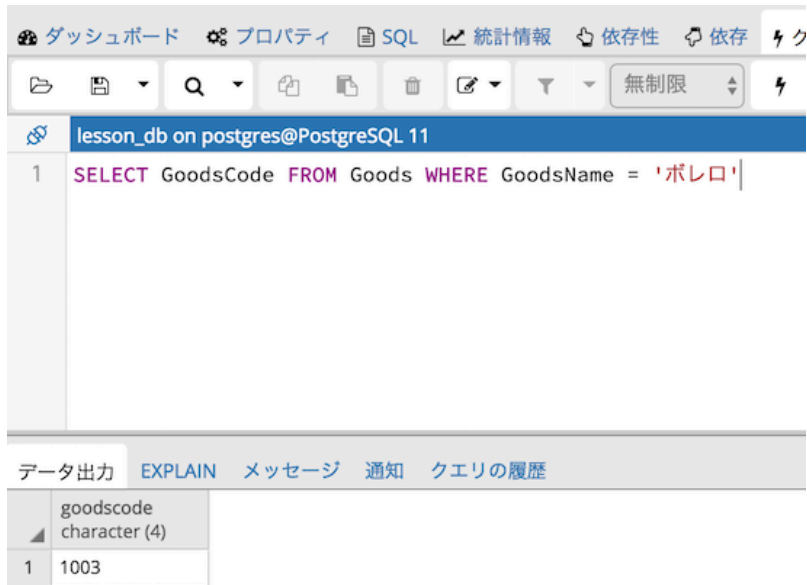
解説

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'
```



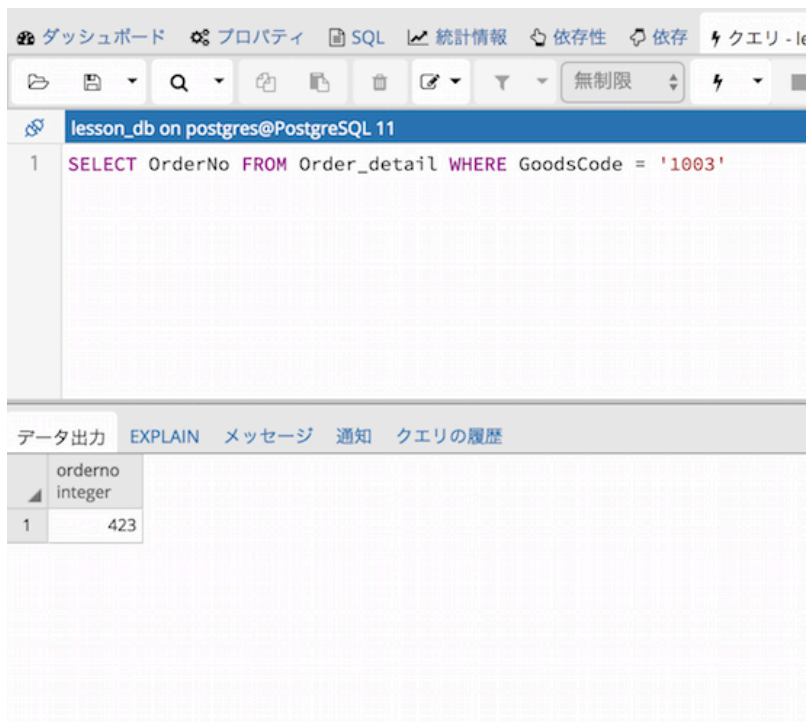
The screenshot shows a database management tool interface. At the top, there's a navigation bar with icons for Dashboard, Properties, SQL, Statistics, Dependencies, and Favorites. Below this is a toolbar with icons for file operations and a search bar. The main area displays the query: `SELECT GoodsCode FROM Goods WHERE GoodsName = 'ボレロ'`. Below the query, there's a tab labeled 'lesson_db on postgres@PostgreSQL 11'. The results are shown in a table with one row:

goodscode
1003

. The column is identified as 'character (4)'.

【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
```



The screenshot shows the same database management tool interface. The query is now: `SELECT OrderNo FROM Order_detail WHERE GoodsCode = '1003'`. The results are shown in a table with one row:

orderno
423

. The column is identified as 'integer'.

↓ 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')
```

The screenshot shows a PostgreSQL query editor interface. The query is as follows:

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

Below the query editor, the results are displayed in a table:

orderno	integer
1	423

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
```

The screenshot shows a PostgreSQL query editor interface. The query is as follows:

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

Below the query editor, the results are displayed in a table:

orderno	integer
1	423

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
```

The screenshot shows a PostgreSQL query editor with the following SQL query:

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

The result set is displayed below the query:

orderno	goodsname
integer	character varying (50)
1	423 ボレロ

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'
```

The screenshot shows a PostgreSQL query editor with the following SQL query:

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

The result set is displayed below the query:

sum	sum
bigint	bigint
1	4000 9500

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
```

ダッシュボード プロパティ SQL 統計情報 依存性 依存 クエリ

lesson_db on postgres@PostgreSQL 11

```
1 SELECT GoodsCode FROM Order_detail
```

データ出力 EXPLAIN メッセージ 通知 クエリの履歴

	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)
```

lesson_db on postgres@PostgreSQL 11

```
1 SELECT * FROM Goods WHERE GoodsCode IN('1002','1003','2001','3001','3002')
```

データ出力 EXPLAIN メッセージ 通知 クエリの履歴

	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)
```


ダッシュボード プロパティ SQL 統計情報 依存性 依存 クエリ

lesson_db on postgres@PostgreSQL 11

```

1 SELECT * FROM Goods WHERE GoodsCode IN
2   (SELECT GoodsCode FROM Order_detail)

```

データ出力 EXPLAIN メッセージ 通知 クエリの履歴

	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)

```

ダッシュボード プロパティ SQL 統計情報 依存性 依存 クエリ

lesson_db on postgres@PostgreSQL 11

```

1 SELECT * FROM Goods TBL1 WHERE EXISTS
2   (SELECT GoodsCode FROM Order_detail TBL2
3     WHERE TBL1.GoodsCode = TBL2.GoodsCode)

```

データ出力 EXPLAIN メッセージ 通知 クエリの履歴

	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)
```

The screenshot shows a PostgreSQL query editor interface. The query is as follows:

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

Below the query editor, the results are displayed in a table:

orderno
integer
1 421

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')
```

```

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

```

データ出力 EXPLAIN メッセージ 通知 クエリの履歴

	orderno integer	subtotal integer
1	421	12800
2	422	8000
3	423	5500

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')
```

```

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

```

データ出力 EXPLAIN メッセージ 通知 クエリの履歴

	orderno integer	sum bigint
1	423	9500
2	421	12800

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
```

ダッシュボード

プロパティ

SQL

統計情報

依存性

依存

クエリ-less

lesson_db on postgres@PostgreSQL 11

1 SELECT SubTotal FROM Order_detail WHERE SubTotal > 5000

2 UNION

3 SELECT SubTotal FROM Order_detail WHERE SubTotal < 6000

データ出力

EXPLAIN

メッセージ

通知

クエリの履歴

	subtotal
	integer
1	4000
2	2000
3	8000
4	12800
5	5500

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
```

The screenshot shows a PostgreSQL query editor with the following SQL query:

```
1 SELECT SubTotal FROM Order_detail WHERE SubTotal > 5000
2 INTERSECT
3 SELECT SubTotal FROM Order_detail WHERE SubTotal < 6000
```

The results tab shows a single row with the value 5500.

subtotal
integer
1 5500

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
```

The screenshot shows a PostgreSQL query editor with the following SQL query:

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

The results tab shows two rows with the values 8000 and 12800.

subtotal
integer
1 8000
2 12800

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

