ＳＱＬ

2015.4.6

１．問合せ

ＳＱＬでは問合せを SELECT 文で行います.

１．１　問合せの基本

（１）基本中の基本

最も簡単な問合せは次のような形式です.

```
SELECT　＊　FROM　<表名>
```

<表名>で示された表（リレーション，テーブル）からすべての行（タプル）を取り出します. FROM の後に検索対象の表の名前 (<表名>) を指定します. SELECT に続く“＊”は，すべての列（属性，カラム）を取り出すという意味です.

なお，ここでは，”SELECT”や”FROM”という語を大文字で記していますが，小文字でも構いません.

テーブル Goods の全データを求める例を以下に示します.

```
SELECT　＊　FROM　Goods
```

次は，特定の列（属性）のみを取り出す場合です.

```
SELECT　<列名>, ・・・　FROM　<表名>
```

<表名>で示された表からすべての行を取り出しますが，<列名>で示された列のみを取り出します．<列名>を指定する場合は「,（カンマ）」で区切ります.

テーブル Goods の属性 gname と price のみを求める例を以下に示します.

```
SELECT　gname, price　FROM　Goods
```

次は，特定の行（タプル）を取り出す場合です．これは，検索条件を指定して行います.

```
SELECT　<列名>, ・・・　FROM　<表名>
  WHERE　<検索条件>
```

<表名>で示された表から，<検索条件>で示された条件を満足する行のみを取り出しま

す．<検索条件>の最も基本的なものは二項演算子を用いたものです．等しいものを求める場合は，<検索条件>を次のように記述します．

```
<列名>　＝　<値>
```

＝のほかに，＜や＞なども指定できます．これらは，（3）で詳しく説明します．また，<値>は，数値の場合はそのまま記述しますが，文字列の場合は「’（シングルクオート)」で文字列の前後をはさむ必要があります．

テーブル Goods のから price が 300 の行（タプル）を求める例を以下に示します．

```
SELECT　＊　FROM　Goods　WHERE　price = 300
```

また，以下は，テーブル Goods のから color が”red”の行（タプル）を求める例です．

```
SELECT　＊　FROM　Goods　WHERE　color = ‘red’
```

以上が問合せの基本中の基本です．

（2）式

（1）では SELECT 句には単純な<列名>しか指定しませんでしたが，ここには式（計算式）を記述することができます．

式は，通常の四則演算（＋，－，＊，／）や剰余（%）が使用できます．また，括弧「(」と「)」でくくることもできます．次の SELECT 文は式を使用した例です．

```
SELECT　price * 2　FROM　Goods
```

（3）検索条件

WHERE 句の<検索条件>に記述できる検索条件について詳しくみてゆきましょう．

(a) 二項演算子

二項演算子には，１．１で示した＝も含めて以下の演算子が使用できます．

・＝　：等しい
・＜＞：等しくない
・＞　：大きい
・＜　：小さい
・＞＝：以上
・＜＝：以下

二項演算子を用いた比較述語の形式は次の通りです.

```
<式>　<二項演算子>　<式>
```

<式>には，単なる<列名>の場合や単なる<値>の場合も含まれます．また，（１）でも述べましたが，<値>が数値の場合はそのまま記述できますが，文字列の場合は「’（シングルクオート)」で文字列の前後をはさむ必要があります.

以下の例は，１０００円以下の商品を求める例です.

```
SELECT　gname, price　FROM　Goods
　WHERE　price <= 1000
```

(b) 論理演算子

(a)で示した比較述語を複数つなげて指定したい場合があります．これは,「かつ」や「または」でつなげるということです．このような論理演算子として以下が使用できます.

・ＡＮＤ：かつ
・ＯＲ　：または
・ＮＯＴ：でない（否定）

以下の例は，１０００円以下のペンを求める例です.

```
SELECT　gname, price　FROM　Goods
　WHERE　price <= 1000 AND gname = ‘pen’
```

ここでは，比較述語を使用した例を示しましたが，以降で説明する様々な述語を含めることもできます．また，さらに複雑な条件は，括弧「()」と論理演算子を用いて記述することができます.

１．２　結合

１．１では，一つの表に対する問合せを扱ってきました．ここでは，２つ以上の表を扱う問合せについて学習しましょう．２つ以上の表は，対象の表を結合して１つの表として扱うことになります.

（１）基本的な結合

１．１で示しましたように FROM 句には検索対象を指定します．２つ以上の表を検索対象にしますから，FROM 句に複数の表を指定することになります．また，一般には，ある条件で複数の表を結合することになります．この結合の条件（結合条件）も検索条件と同様に WHERE 句に指定します. さらに, 異なる表の中に同じ列名があり得ますので,

この場合は，上記のように，<列名>を<表名>で修飾する必要があります.

```
SELECT　<表名>.<列名>, ・・・　FROM　<表名>, ・・・
  WHERE　<結合条件>
```

以下の例は，商品名と在庫を求める例です.

```
SELECT　Goods.gname, Stock.qty　FROM　Goods,　Stock
  WHERE　Goods.gno = Stock.gno
```

この例では，商品番号（gno）が等しいという条件で結合しています．指定する二項演算子は＝のみではなく他のものも使用できます.

また，無条件に複数の表のタプルを突き合わせることもできます.

```
SELECT　<表名>.<列名>, ・・・　FROM　<表名>, ・・・
```

これは，無条件に複数の表のタプルを突き合わせることになります．２つの表を無条件に結合する場合，おのおのの表のタプルの数をそれぞれＮとＭとすると，結合の結果できる表はＮ＊Ｍ行のタプルを持つことになります.

同じ結合を別の構文でも書くことができます.

```
SELECT　<表名>.<列名>, ・・・
  FROM　<表名>, ・・・　ON　<結合条件>
```

例えば，次のようになります.

```
SELECT　Goods.gname, Stock.qty
  FROM　Goods, Stock　ON　Goods.gno = Stock.gno
```

また，この通常の結合を，内結合（内部結合）と呼ぶこともありますが，これを陽に記述して以下のようにも書くことができます.

```
SELECT　<表名>.<列名>, ・・・
  FROM　<表名>　INNER JOIN ・・・　ON　<結合条件>
```

例えば，次のようになります.

```
SELECT  Goods.gname, Stock.qty
  FROM  Goods INNER JOIN Stock  ON  Goods.gno = Stock.gno
```

（2）相関名

結合では，列名の前に表名をつけなければならないといけない場合が多くあります．このような場合に，表に別名（変数名とも考えられる）を付けると便利なことがあります．これが相関名です．

```
SELECT  <相関名>.<列名>, ・・・  FROM  <表名>  <相関名>, ・・・
```

前記の例を相関名を使うと次のように書けます．

```
SELECT  g.gname, s.qty  FROM  Goods g,  Stock s
  WHERE  g.gno = s.gno
```

次のようにも書けます．

```
SELECT  g.gname, s.qty
  FROM  Goods g  INNER JOIN  Stock s  ON  g.gno = s.gno
```

１．３　順序化

検索を行っていると，問合せの結果を何らかの列に着目して並べ替えて表示してほしいということがあります．今までのところ１．１で説明した SELECT 文でもあまり問題はないように思えますが，実は，検索結果の順序が毎回同じであるという保証はありません．実際，使用するデータベース管理システムによっては，主キーの順番で表示されたり，格納順で表示されたり，そのようなものとは全く関係のない順序で表示されたりします．どのようなデータベース管理システムを使用しても同じ順序で検索結果を得るためには，どのような順序で並べるのかを指定しておく必要があるのです．

検索結果の順序付けは ORDER BY 句によって行います．

```
SELECT  <列名>, ・・・  FROM  <表名>
  WHERE  <検索条件>
  ORDER BY  <列名>  {ASC | DESC}, ・・・
```

順序の優先順位が高い列から順番に指定します．値が小さいものから大きいものへの昇

順を指定するには ASC とします．逆に，値が大きいものから小さいものへの降順を指定するには DESC とします．何も指定しないと ASC を指定したことになります．

テーブル Goods から price の大きいものの順に（降順に）データを求める例を以下に示します．

```
SELECT ＊ FROM Goods ORDER BY price DESC
```

次週以降の説明では，説明上必要である場合を除いて ORDER BY 句を記述しませんが，通常のＳＱＬ文，特に，プログラム中から発行するＳＱＬ文では ORDER BY 句を指定しておくのが良いでしょう．

１．４　様々な述語

検索条件に記述する様々な述語について学びましょう．

(a) LIKE 述語

文字列のパターン照合を行うための述語が LIKE 述語です．

```
<列名> LIKE <パターン指定>
```

<パターン指定>には，以下の特殊な意味を表す文字を含めることができます．

・％：任意の文字列（長さ０を含む）

・＿：任意の一文字

以下の例は，文字列「oo」を含む商品名を求める例です．

```
SELECT gname FROM Goods
  WHERE gname LIKE ‘%oo%’
```

文字列データに対する<パターン指定>は文字列ですので，「’（シングルクオート）」で前後をはさむ必要があります．

名前が文字「p」で始まる商品を求めるには以下のようにします．

```
SELECT gname FROM Goods
  WHERE gname LIKE ‘p%’
```

同様に，名前が文字「r」で終わる商品を求めるには以下のようにします.

```
SELECT  gname  FROM  Goods
  WHERE  gname  LIKE  '%r'
```

名前が文字「p」で始まり３文字の名前の商品を求めるには以下のようにします.

```
SELECT  gname  FROM  Goods
  WHERE  gname  LIKE  'p_ _'
```

(b) BETWEEN 述語

値がある範囲に存在するかを条件にしたいことが良くあります．このための述語が BETWEEN 述語です.

```
<式>  BETWEEN  <開始値>  AND  <終了値>
```

以下の例は，１００円以上２００円以下の値段の商品を求める例です.

```
SELECT  gname, price  FROM  Goods
  WHERE  price  BETWEEN  100  AND  200
```

BETWEEN 述語は以下の条件式と等価です.

```
<式>  >=  <開始値>  AND  <式>  <=  <終了値>
```

(c) NULL 述語

値がＮＵＬＬかどうかの検査を行うための述語が NULL 述語です.

```
<列名>  IS  NULL
```

例えば，色の値が NULL の商品名を求めるには以下のようにします.

```
SELECT  gname  FROM  Goods
  WHERE  color  IS  NULL
```

値がＮＵＬＬであるということと，文字列が空文字列「""」であるということは意味が

異なりますので注意が必要です. 空文字列も空文字列という値があることになりますので値がＮＵＬＬではありません.

また，NULL でないものを求めるときは，以下のように書くことができます.

```
SELECT  gname  FROM  Goods
  WHERE  color  IS  NOT  NULL
```

ここでは，色の値がＮＵＬＬでないものを求めています.

(d) IN 述語

値がある値かという検査を行いたいことも良くあります．このための述語が IN 述語です．IN 述語は以下の形式です.

```
<列名>  IN  ( <値>, ・・・ )
```

<値>で示された値の場合，この述語は真となります.
例えば，色の値が赤または青の商品の名前を求めるには以下のようにします.

```
SELECT  gname  FROM  Goods
  WHERE  color  IN  ('red', 'blue')
```

そうでないものを求めるときは，以下のように書くことができます.

```
SELECT  gname  FROM  Goods
  WHERE  color  NOT  IN  ('red', 'blue')
```

ここでは，色が赤や青でないものを求めています.

１．５　重複排除

ここまでの学習で問合せの結果に重複のあることに気付いた人もいることと思います．これで問題のないこともありますが，重複は排除してほしい場合もあります．このときには，SELECT 句に「DISTINCT」を付けます.

```
SELECT  DISTINCT  <列名>, ・・・  FROM  <表名>
```

例えば，商品名を重複なく求めるには次のようにします．

```
SELECT　DISTINCT　gname　FROM　Goods
```

１．６　集約関数とグループ化

次に，集約とグルーピングを行う処理について学習しましょう．

（１） 集約関数

以下に示す集約関数が使えます．

- COUNT：個数
- AVG：平均
- SUM：合計
- MIN：最小値
- MAX：最大値

次に示す例は，鉛筆の種類の数を求める例です．

```
SELECT　count(*)　FROM　Goods
  WHERE　gname = 'pencil'
```

count の中に＊を記述するとタプル数を求めることになります．count の中に列名を指定することもできます．この場合，その列がＮＵＬＬのものはカウントに含まれません．以下の例は，色がＮＵＬＬでない商品の数を求める例です．

```
SELECT　count(color)　FROM　Goods
```

残りの集約関数はその名の通りの働きをします．

（２） グループ化

ある列の値でグループ化したいということも良くあります．これは GROUP BY 句を用いて行います．

```
SELECT　<式>, ・・・　FROM　<表名>
    GROUP BY　<列名>, ・・・
```

GROUP BY で指定した<列名>の順で優先されてグループ化されます．

商品を商品名でグループ化し，各商品種別内の商品数と価格の平均を求めるには次のようにします．

```
SELECT　gname, count(*), avg(price)　from Goods
  GROUP BY gname
```

商品名（gname）でグループ化しますから，SELECT 句にはグループ化対象の列（この場合は gname）のほかには集約関数（とこれを含む式）しか書けないことに注意してください.

（3） グループ化の条件

グループ化する際に条件を付けたいことがあります．これは HAVING 句を用いて行います.

```
SELECT　<式>, ・・・　FROM　<表名>
    GROUP BY　<列名>, ・・・
    HAVING　<グループ化条件>
```

<グループ化条件>には，SELECT 句と同様に，グループ化対象の列と集約関数（とこれを含む式）が記述できます.

商品名が'pencil'のもののみ商品を商品名でグループ化し，各商品種別内の商品数と価格の平均を求めるには次のようにします.

```
SELECT　gname, count(*), avg(price)　from Goods
  GROUP BY gname
  HAVING　gname = 'pencil'
```

また，商品数が１個を超えるもののみにするには以下のようにします.

```
SELECT　gname, count(*), avg(price)　from Goods
  GROUP BY gname
  HAVING　count(*) > 1
```

さらに，最も高い価格が２００円未満のものだけにするには以下のようにします.

```
SELECT　gname, count(*), avg(price)　from Goods
  GROUP BY gname
  HAVING　max(price) < 200
```

２．更新

データを更新するＳＱＬ文には，INSERT 文（挿入），DELETE 文（削除）と UPDATE 文（修正）があります.

２．１　挿入

行（タプル）をテーブルに挿入するには INSERT 文を使用します．最も基本的な書式は以下のようなものです.

```
INSERT INTO　<表名>　VALUES ( <値>, ・・・ )
```

<値>は，<表名>で示されたテーブルの属性の定義順に指定します.

テーブル Goods に新たなタプルを挿入するには次のようにします.

```
INSERT INTO　Goods　VALUES (‘G21’, ‘PC’, ‘white’, 100000)
```

<値>を省略する場合は，”NULL”と記述します.

色(color)の値を省略してタプルを挿入する例を次に示します.

```
INSERT INTO　Goods　VALUES (‘G22’, ‘bag’, NULL, 10000)
```

“NULL”と記述できるのは，テーブルの定義時に「省略可」とした属性のみです.

このほかに，属性を指定してタプルを挿入することもできます.

```
INSERT INTO　<表名>　( <列名>, ・・・ )　VALUES ( <値>, ・・・ )
```

<値>は，<列名>で示された属性の順で指定します.

例えば，テーブル Goods に商品番号(gno)と商品名(gname)の値のみを指定して新たなタプルを挿入するには次のようにします.

```
INSERT INTO　Goods (gno, gname)　VALUES　(‘G23’, ‘PC’)
```

２．２　削除

行（タプル）をテーブルから削除するには DELETE 文を使用します.

```
DELETE　FROM　<表名>
  WHERE　<条件式>
```

<表名>で示されたテーブルから，<条件式>を満足するタプルを削除します.

テーブル Goods から価格(price)が 300 円未満の商品を削除するには次のようにします.

```
DELETE　FROM　Goods　WHERE　price < 300
```

テーブルから全タプルを削除するには以下のようにします.

```
DELETE　FROM　<表名>
```

テーブル Goods の全タプルを削除するには以下のようにします.

```
DELETE　FROM　Goods
```

２．３　修正

行（タプル）を修正するには UPDATE 文を使用します.

```
UPDATE　<表名>　SET <列名> = <修正値>, ・・・
  WHERE　<条件式>
```

<表名>で示されたテーブルの<条件式>を満足するタプルの<列名>を持つ属性の値を<修正値に>修正します. WHERE 節を省略すると, テーブル中の全タプルが修正対象になります.

テーブル Goods の商品番号(gno)が”G22”の商品の価格(price)を 2 倍にし，色(color)を”green”に修正するには次のようにします.

```
UPDATE　Goods　SET　price = price * 2, color = 'green'
    WHERE　gno = 'G22'
```

テーブル Goods の全商品の価格(price)を 2 倍にするには次のようにします.

```
UPDATE　Goods　SET　price = price * 2
```

３．定義

テーブルを作成するためのＳＱＬ文として CREATE　TABLE 文があります．また，テーブルを削除するためのＳＱＬ文として DROP　TABLE 文があります．

３．１　テーブル作成

テーブルを作成するには CREATE　TABLE 文を使用します. 最も基本的な書式を以下に示します.

```
CREATE　TABLE　<表名>　（<列名> <データ型名> [<列制約>], ・・・ ）
```

<表名>で示されたテーブルが新たに作成されます．このテーブルは，<列名>で示された属性を持ちます．<表名>は，データベース内で一意の名前でなければなりません．<列制約>には，省略不可を示す”NOT　NULL”等を指定できます．<データ型名>には，表１に示すデータ型が使用できます（このほかにもあります）.

表１　データ型

データ型	意　　味
CHAR(n)	固定長文字列，ｎに長さ（１～２５４）を指定
VARCHAR(n)	可変長文字列，ｎに長さを指定（１～４０００）
INTEGER または INT	整数（-2147483648～+2147483647）
SMALLINT	整数（-32768～+32767）
DECIMAL(p,s)	有効桁数 p，位取り s の数値（位取りは小数点の右の桁数）
FLOAT または REAL	単精度浮動小数点　例 8.934000e+03
DOUBLE PRECISION	倍精度浮動小数点　例 8.99987654311320e+01

属性として，学生番号(sno)（整数型で省略不可），氏名(sname)（24 バイト固定長文字列），住所(address)（127 バイト可変長文字列）を持つテーブル Student を作成するには次のようにします.

```
CREATE　TABLE　Student ( sno　integer　NOT　NULL,
                         sname　char(24),
                         address　varchar(127) )
```

３．２　テーブル削除

テーブルを削除するには DROP　TABLE 文を使用します.

```
DROP　TABLE　<表名>
```

テーブル Student を削除するには次のようにします.

```
DROP　TABLE　Student
```

４．ビュー定義

３．で述べたテーブルは実際に存在するテーブルで，実テーブルといいます．これに対して，他のテーブルをもとにした仮想的なテーブルをビューテーブル（以降，簡単のためにビューと記します）といいます．ビューは CREATE　VIEW 文を用いて定義できます．また，ビューを削除するには DROP　VIEW 文を用います.

４．１　ビュー定義

ビューを定義するには CREATE　VIEW 文を使用します．最も基本的な書式を以下に示します.

```
CREATE　VIEW　<ビュー名>
  AS　<SELECT 文>
```

<ビュー名>で示されたビューが新たに作成されます．このビューは，AS の後に記述する<SELECT 文>で得られる結果を内容とする仮想的なテーブルです.

以下の例は，１０００円以下の商品の商品名と価格からなるビューを定義する例です.

```
CREATE　VIEW　CheapGoods
  AS　SELECT　gname, price　FROM　Goods　WHERE　price <= 1000
```

ビューCheapGoods の列名は，もとの Goods と同じ，gname と price になります．テーブル Goods の中の price が 1000 以下のタプルのみとなります.

列名を，もとの Goods と異なるものにすることもできます.

```
CREATE　VIEW　CheapGoods2　(cheap_gname, cheap_price)
　AS　SELECT　gname, price　FROM　Goods　WHERE　price <= 1000
```

この例では，cheap_gname と cheap_price という列名となります.

ビューは，実テーブルのみではなく，ビューをもとに定義することもできます.

```
CREATE　VIEW　VeryCheapGoods
　AS　SELECT gname, price FROM CheapGoods WHERE price <= 100
```

ビューChaepGoods をもとにしてビューVeryCheapGoods を定義しています.

ビューの定義に使用する SELECT 文は，任意の SELECT 文が記述できます．例えば，結合を指定することもできます.

```
CREATE　VIEW　StockInfo
　AS　SELECT　Goods.gname, Stock.qty　FROM　Goods,　Stock
　　WHERE　Goods.gno = Stock.gno
```

テーブル Goods と Stock を結合してできる表をビューとしています. Stock には商品番号（gno）しか格納されていませんでしたが，StockInfo には，商品名（gname）と在庫数（qty）が格納されているとして扱えます．結合の対象は，実テーブルとビューテーブルを混在させることもできますし，複数のビューテーブルを結合することもできます.

４．２　ビュー削除

ビューを削除するには DROP　VIEW 文を使用します.

```
DROP　VIEW　<ビュー名>
```

<ビュー名>で指定されたビューが削除されます．ただし，ビューのもととなったテーブルには何の影響もありません.

５．複雑な問合せ

５．１　結合

（１） 自己結合

場合によっては，自分自身の表と結合しなければならないことがあります．これを自己結合と呼びます．

例えば，人の名前とその子どもの名前が格納されている表 person をもとにして孫を求める問合せは以下のように記述します．

```
SELECT　p.pname, c.child_name　FROM　person p, person c
  WHERE　p.child_name = c.pname
```

このように自己結合の場合は相関名を必ず使うことになります．

（２） 外結合

これまでの結合では，結合する相手がいないタプルは問合せ結果には現れません．しかし，場合によっては結合する相手がいない場合でも問合せ結果に残したいことがあります．この場合に使用するのが外結合（outer join）です．外結合には，左側の表の行をすべて含む左外結合（left outer join），右側の表の行をすべて含む右外結合（right outer join），両方の行をすべて含む完全外結合（full outer join），ならびに，単に２つの表の和をとるような和結合（union join）があります．

左外結合の構文は次のようになります．

```
SELECT　<表名>.<列名>, ・・・
  FROM　<表名>　LEFT OUTER JOIN ・・・　ON　<結合条件>
```

左外結合の例は，次のようになります．

```
SELECT　p.pname, c.child_name
  FROM person p LEFT OUTER JOIN person c ON p.child_name = c.pname
```

内結合と比較すると違いが分かると思います．

右外結合と完全外結合も同じような構文で記述できます．和結合の場合は，ON の後の結合条件を指定しません．

５．２　副問合せ

SELECT 文の WHERE 句の中にさらに SELECT 文を書くことができます．これを副問合せまたは副照会(subquery)と呼びます．

（１） 単純な副問合せ

IN 述語では，検索対象の値の集合を指定できました．これを問合せ（すなわち，SELECT 文）で指定することができます．

例えば，在庫が１個のみの商品の商品名を副問合せを用いて記述すると以下のようになります．

```
SELECT　gname FROM Goods
  WHERE gno IN (SELECT gno FROM Stock WHERE qty = 1)
```

副問合せの（すなわち括弧の中の）SELECT 文で求まる結果が副問合せに置き換わると考えると分かりやすいでしょう．なお，この問合せは結合を用いても記述することができますが，副問合せを含む問合せが結合を用いて記述できるとは限りません．

（２） 相関問合せ

（１）では，副問合せと外側の問合せ（主問合せ，主照会）には関係がありませんでした．ここでは，副問合せと主問合せに相関を持つものについて示します．

例えば，種別ごとの平均価格よりも高い商品を求める問合せは以下のように書けます．

```
SELECT　g.gname, g.price　FROM　Goods　g
  WHERE　g.price >= (SELECT avg(price)　FROM　Goods　ag
                                          WHERE g.kind = ag.kind)
```

６．制御

（１）トランザクション開始

トランザクションの開始は以下の文で行います.

```
BEGIN　TRANSACTION
```

```
BEGIN
```

（２）書き込み終了

操作の結果を DB に反映させて終了する場合は以下の文で行います.

```
COMMIT　TRANSACTION
```

```
COMMIT
```

（３）破棄終了

操作の結果を DB に反映させないで終了する場合は以下の文で行います.

```
ROLLBACK　TRANSACTION
```

```
ROLLBACK
```

（４）終了

トランザクション中の操作が更新を含まない場合は以下の文で終了できます. 更新を含む場合も終了できますが, COMMIT となるか ROLLBACK となるかは設定やシステムによって異なることが多いので注意した方が良いでしょう.

```
END　TRANSACTION
```

```
END
```