SQL によるデータ照会の例題

はじめに

SQL（Structured Query Language，構造化照会言語）は，リレーショナル・データベースを操作するための言語である。つまり，データベースを構築したり，データを探索したり，データを更新したりするのに使用する言語である。

ここでは，SQL によるデータ探索の分野を学ぶことにしよう。

リレーショナル・データベースからデータを探索するための理論的基礎を与えているのは，集合演算としてのリレーショナル代数である。そしてリレーショナル代数を実現することのできる言語の１つが，SQL である。

しかしながら，SQL はリレーショナル代数の８つの演算をそれぞれ直接に実現するようなコマンド群として存在するわけではない。つまり１対１に対応するわけではない。リレーショナル代数を SQL で実現するには， SQL の構文にしたがってプログラミングをする必要がある。

以下は，SQL プログラミングの事例を示したものである。サンプルデータを用いて練習できるようになっている。

SQL の基本構文

SQL ステートメントによる問合せ（データ照会）の基本構文は，次のような形式である。なお[ ]内はなくてもよい。

SQL ステートメント	意味
SELECT 列名リスト	xxx 列を抜き出して表示しなさい
FROM テーブル名リスト	xxx テーブルから探索して
[WHERE 探索条件]	探索条件が真(true)になる行を選択して
[GROUP BY 列名リスト]	xxx 列の値でグルーピングして
[HAVING 探索条件]	探索条件が真(true)であるグループを選択して

これらは SQL ステートメントの基本要素で， SELECT 句，FROM 句，WHERE 句，GROUP BY 句，HAVING 句という。

SQL による問合せの例題

以下の例題は，サンプル用のデータベース・ファイル 'rdb2.mdb'を用いて実習できる。

また以下の SQL ステートメントは，MS-Access の SQL に従っている。 JIS 規格の SQL と違う点がある。拡張されている。

例題 01　すべての列を表示する

```
SELECT *
FROM   C1;
```

テーブル C1 から，すべての行について，すべての列を抜き出して表示する。
*は，すべての列を意味する総称文字である。
文の終わりの;(セミコロン)はなくてもよい。
文の途中に挿入する空白は，半角の空白でなければならない。

例題 02　列を抜き出して表示する

```
SELECT 顧客，担当社員
FROM   B1;
```

テーブル B1 から，すべての行について，
列「顧客」と「担当社員」を抜き出して表示する。
列名が複数あるときは，,(カンマ)で区切る。

例題 03　マルチ集合

```
SELECT スーパー
FROM   C1;
```

テーブル C1 から，すべての行について，
列「スーパー」を抜き出して表示する。
列「スーパー」の値であるスーパーの名称が表示されるが，
このときいくつかの行で値が重複していても，
そのまま重複した行のまま表示される。
このように重複した値をとる集合をマルチ集合(multi-set)という。
なお，重複しているというのは，行を集合の元とみて，
２つ以上の行の間ですべての列の値が等しいことをいう。
１つでも値の違う列があれば，
それらの行は重複しているとはいわない。

例題 04　DISTINCT による修飾

```
SELECT DISTINCT スーパー
FROM   C1;
```

テーブル C1 から，すべての行について，
列「スーパー」を抜き出して表示する。
このとき重複のないスーパー名の集合として抜き出す。
SELECT 句で DISTINCT という修飾語をつけると，
マルチ集合でなくなる。

例題 05　文字定数

```
SELECT 商品，単価円，'円'
FROM   B2;
```

テーブル B1 から，すべての行について，
列「商品」と「単価円」を抜き出して表示する。
このとき，文字「円」を表示する列を追加する。
このように列の一部として，一定の文字列（文字定数）を表示させるには，
文字を'(single quotation)ではさんで，列リストに加える。
このとき列名は自動的に設定される。
自動的に設定される列名に換えて，列名を指定したいときは，

```
SELECT 商品，単価円，'円' AS 単位
FROM   B2;
```

のように，
　AS 新列名
を付け加えればよい。新列名は，'(single quotation)ではさまなくてもよい。
なお，一定の数値を数値として表示させるときは，
'(single quotation)ではさまなくてもよい。
はさんだときは文字としての扱いになる。

（注）JIS 規格 SQL では，半角英数字の文字定数は，
　　'ABC123'
と表し，漢字による文字定数は，
　　N'麗澤かんじ'
のように先頭に N をつけて表記しなければならない。

例題 06　計算式

```
SELECT 最寄り駅，通勤手当円/1000，'千円'
FROM   通勤手当早見表;
```

テーブル「通勤手当早見表」から，すべての行について，
列「最寄り駅」を抜き出して表示するとともに，
「通勤手当円」を 1000 で割った値と，文字定数「千円」を表示する。
この例のように SELECT 句の列名リストに計算式を書くことができる。
計算式からなる列は新たな列とみなされて，
自動的に設定された列名が表示される。
計算式で使用することのできる主な算術演算子として次のものがある。

算術演算子	使用例	意味（X と Y は列名か数値である）
* 掛算	X * Y	X に Y を掛ける。
/ 割算	X / Y	X を Y で割る。
+ 足算	X + Y	X に Y を足す。
- 引算	X - Y	X から Y を引く。
^ べき乗	X ^ Y	X を Y 乗する。
Mod 余り	X Mod Y	X を Y で割った余り。
\ 商	X \ Y	X を Y で割った商。

（注）JIS 規格 SQL では，^，Mod，\の演算子はない。

例題 07　WHERE 句と探索条件及び比較述語

```
SELECT 社員名
FROM   社員通勤表
WHERE  通勤手段 = '鉄道';
```

テーブル「社員通勤表」から，
通勤手段が「鉄道」である行を選択して，
その行の社員名を表示する。
つまり鉄道で通勤している社員をリストアップする。
行を選択するための探索条件を定義している 3 行目を WHERE 句という。
一般に WHERE 句の探索条件は，１つ以上の述語（条件式のこと）を，
AND，OR，NOT で組み合せてできた論理式である。
述語（条件式）は，探索条件を構成する最小の単位で，
真(true)か偽(false)か不定(Null，ナル)のどれかの値をとる。
１つ以上の述語が AND，OR，NOT で結合された探索条件全体は，
論理式として評価され最終的に真か偽か不定のどれかの値をとる。
この例では，
探索条件は「通勤手段 = '鉄道'」という１つの述語だけからできている。
この述語は，
ある行の列「通勤手段」の値が'鉄道'という値に等しいとき，
真(true)の値をとる。
２つの間の大小関係を比較する述語を比較述語という。
述語の種類は他にもある。
比較述語で使用される=のような演算子を比較演算子という。
比較演算子には次のものがある。

比較演算子		比較述語	意味（X と Y は列名か数値か文字定数である）
=	等号演算子	X = Y	X と Y が等しい。
<	未満演算子	X < Y	X が Y より小さい。
>	超演算子	X > Y	X が Y より大きい
<>	不等号演算子	X <> Y	X と Y とが等しくない。
>=	以上演算子	X >= Y	X が Y 以上である。
<=	以下演算子	X <= Y	X が Y 以下である。

(注)文字定数は'(single quotation)ではさまなければならない。

データ処理の流れとしていえば，
テーブルの行ごとに探索条件が評価され，
真の値をとる行だけが選択される。

例題 08　AND，OR，NOT

```
SELECT 最寄り駅，通勤手当円，'円'
FROM   通勤手当早見表
WHERE  通勤手当円 >= 100000
AND    通勤手当円 <= 150000;
```

テーブル「通勤手当早見表」から，
列「通勤手当円」の値が 100000 以上でかつ 150000 以下の行を選択して，
列「最寄り駅」と「通勤手当円」と文字定数'円'を表示する。
この例では，２つの述語（条件式）が AND で結合して探索条件になっている。
AND の書き方は，
　　述語 1 AND 述語 2
である。述語 1 が真でかつ述語 2 が真のとき，真の値をとる。
OR の書き方は，
　　述語 1 OR 述語 2
である。述語 1 が真かまたは述語 2 が真のとき，真の値をとる。
NOT の書き方は，
　　NOT(述語 1)
である。述語 1 が真なら偽，述語 1 が偽なら真の値をとる。

AND，OR，NOT による論理演算の値

AND	真	偽	不定
真	真	偽	不定
偽	偽	偽	偽
不定	不定	偽	不定

OR	真	偽	不定
真	真	真	真
偽	真	偽	不定
不定	不定	不定	不定

NOT	
真	偽
偽	真
不定	不定

例題 09　BETWEEN 述語

```
SELECT 最寄り駅，通勤手当円，'円'
FROM   通勤手当早見表
WHERE  通勤手当円 BETWEEN 100000 AND 150000;
```

これは，上の例題 ex08 の WHERE 句を BETWEEN 述語を用いて書き換えたものである。
「通勤手当円 BETWEEN 100000 AND 150000」の部分を BETWEEN 述語という。
BETWEEN 述語は次のような書き方をする。
　　値式 1 BETWEEN 値式 2 AND 値式 3
値式には，列名，数値，または計算式がくる。
BETWEEN 述語は，値式 1 が値式 2 以上でかつ値式 3 以下のとき真の値をとる。
それ以外は偽の値をとる。
BETWEEN 述語の中の AND は，
述語を結合するときの論理演算子としての AND とは違って，
BETWEEN 述語の一部である。
また「通勤手当円 NOT BETWEEN 100000 AND 150000」と書くと，
否定形の BETWEEN 述語になる。
「通勤手当円」が「100000 以上かつ 150000 以下」でないとき，真の値をとる。
否定形の BETWEEN 述語は次のような書き方をする。
　　値式 1 NOT BETWEEN 値式 2 AND 値式 3
値式 1 が「値式 2 以上でかつ値式 3 以下」でないとき真の値をとる。
このときの NOT は，BETWEEN 述語の一部としての NOT である。
否定形の BETWEEN 述語は，
　　NOT (値式 1 BETWEEN 値式 2 AND 値式 3)
と書いたのと結果は同じである。
このときの NOT は，
述語を結合して探索条件を作るときの論理演算子としての NOT である。

例題 10　LIKE 述語と文字列のパターン・マッチング

```
SELECT 社員名
FROM   社員通勤表
WHERE  社員名 LIKE '*子';
```

テーブル「社員通勤表」から，
列「社員名」の値が'子'で終わる行を選択して，
列「社員名」を表示する。
キーワード LIKE で構成される述語を LIKE 述語という。
LIKE 述語は，次のような書き方をして文字列のパターン・マッチングを行う。
　　列名 LIKE '文字パターン'
列がとる値（文字列）が，
ある文字パターンに一致しているとき真の値をとる。
文字パターンでは，
次のような特殊文字を使用して特定の文字パターンを意味することができる。
（注）JIS 規格 SQL はこれとは全く異なる。

特殊文字	意味
?	任意の 1 文字
*	任意の数の文字
#	任意の 1 文字の数字 (0-9)
[charlist]	文字リスト charlist に指定した文字の中の 1 文字

（例）[a-z]とするとアルファベット小文字のどれか 1 つを意味する。
[高鷹隆]とすると高，鷹，隆のどれか 1 つを意味する。
[!charlist]　文字リスト charlist に指定した文字以外の任意の 1 文字
（例）[!a-c]とすると，a,b,c 以外の 1 文字を意味する。

これらのうち，"#" を除くすべての文字パターンでは，
2 バイト文字（全角文字）も 1 文字と数えて文字列比較を行う。
"#" には，1 バイト（半角）の数字だけが一致する。
1 個以上の文字のリスト（charlist）を角かっこ（[ ]）
で囲んで文字パターンに指定すると，
その中のいずれかの文字と一致するかどうかを比較することができる。

例	説明
社員名 LIKE '*子'	「社員名」が'子'で終わるとき真。'鳩山麗子'や'橋本龍子'は真。'加藤子羊'は偽。
社員 CODE LIKE 'S4???'	「社員 CODE」が'S4'で始まる 5 桁の文字のとき真。'S4'で始まっていても 5 桁でないときは偽。
社員名 LIKE '*山*'	「社員名」の最初でも途中でも最後でも，どこかに'山'を含んでいれば真。
社員 CODE LIKE 'S[4-6][1-3]??'	「社員 CODE」が'S'で始まっていて， かつ左 2 桁目が 4～6 の数字の 1 つで， かつ左 3 桁目が 1～3 の数字の 1 つで， かつ全体が 5 桁の文字なら真。

例題 11　IN 述語

```
SELECT 住所 CODE，最寄り駅
FROM   通勤手当早見表
WHERE  住所 CODE IN ( 'J108', 'J116', 'J123');
```

テーブル「通勤手当早見表」から，
列「住所 CODE」の値が，'J108'か'J116'か'J123'のどれかをとる行を選択して，
列「住所 CODE」と「最寄り駅」を表示する。
IN 述語は，離散的に指定した複数の値のうちどれかに一致するものを選ぶとき使用する。
IN 述語の書き方
　値式 IN (値 1，値 2，...)
値式とは，列名か計算式である。値 1 や値 2 とは，数値か文字定数である。
IN 述語は，値式が，値 1，値 2，...のどれかに一致するとき真の値をとる。
（例）
　住所 CODE IN ('J108', 'J116', 'J123')
　通勤手当 IN (100000, 125000, 56000)
文字定数は，'(single quotation)ではさまなければならない。
数値は，そのまま記述する。
IN 述語の否定形は，
　値式 NOT IN (値 1，値 2，...)
である。これは，
　NOT(値式 IN (値 1，値 2，...))
としても同じ結果になる。

例題 12　集合関数 SUM

```
SELECT  SUM(数量) AS 柏クッキー売上総数量
FROM    B1
WHERE   売上商品='柏クッキー';
```

テーブル B1 から，
売上商品が'柏クッキー'の行を選択して，
選択した行全体について，
列「数量」の値の合計を表示する。
つまり'柏クッキー'の売上数量の合計を求める。
なお合計を表示する列には「柏クッキー売上総数量」という列名を設定している。

集合関数は，選択した行全体について，
引数で指定した列を用いて何か計算を行う関数である。
この例の SUM 関数は，WHERE 句で選択された行の列「数量」の値の合計をとっている。

例題 13　集合関数 COUNT

```
SELECT COUNT(*) AS 自動車通勤者数
FROM   社員通勤表
WHERE  通勤手段 = '車';
```

テーブル「社員通勤表」から，
車で通勤している社員の数を求める。
COUNT 関数は，選択された行の件数をカウントする。
引数には列名を指定しないで，単に*を指定することが多い。
（注）JIS 規格 SQL では，COUNT(DISTINCT 社員名)のように
　　DISTINCT を付けた列名を指定すると，
　　社員名のうち重複を除いた名称の件数が返る。
　　MS-Access ではこの表現は使用できない。

例題 14　GROUP BY 句によるグルーピング

```
SELECT   売上商品，SUM(数量) AS 数量合計，COUNT(*) AS 売上回数
FROM     B1
GROUP BY 売上商品;
```

テーブル B1 から，売上商品の名称でグルーピングして数量合計と売上回数とを求める。
GROUP BY 句は，指定した列の値が同じ行どうしを集めてグルーピングする。
ここでは売上商品の列の商品名でグルーピングしている。
グルーピングされた行全体は，
あたかも１つの行であるかのように，いくつかの属性を持つことになる。
その場合の属性とは，
もとの行の列の値か，またはそれから集合関数で計算され得るもので，
グループの属性として意味のあるものである。
SELECT 句の列リストには，そうしたグループの属性が並ぶ。
上の例のように列「売上商品」の場合は，
グループの属性としてグループ内で共通している商品名が表示される。
数量の合計もグループの属性として意味がある。しかし，

単に列「数量」を指定してもグループとしての属性にならないので実行できない。

例題 15 HAVING 句によるグループの選択

```
SELECT   売上商品，SUM(数量) AS 数量合計，COUNT(*) AS 売上回数
FROM     B1
GROUP BY 売上商品
HAVING   売上回数 >= 2;
```

テーブル B1 から，売上商品の名称でグルーピングして，
売上回数が 2 以上のグループを選択して，
数量合計と売上回数とを求める。
HAVING 句は，GROUP BY 句でできるグループのうちから，
特定のグループを選択するための探索条件を指定する。
このときの探索条件の書き方は，WHERE 句の探索条件の書き方と同じである。

例題 16 結合（join）：２つのテーブルを結合する(1)

```
SELECT 社員通勤表.社員名，
       通勤手当早見表.最寄り駅，
       通勤手当早見表.通勤手当円
FROM   社員通勤表，通勤手当早見表
WHERE  社員通勤表.住所 CODE = 通勤手当早見表.住所 CODE;
```

テーブル「社員通勤表」とテーブル「通勤手当早見表」とから，
２つのテーブルの間で住所 CODE が一致する行を選択して，
列「社員名」，「最寄り駅」，「通勤手当円」を表示する。
このように２つ以上のテーブルから行と列とを選択して
１つのリレーションを作成することを結合という。
リレーショナル・データベースの最も特徴的な利用方法である。
SELECT 句の列リストで列名を書くときは，
「社員通勤表.社員名」のように，
列名「社員名」の前にテーブル名「社員通勤表」をつけて修飾する。
間には区切りの.(ピリオド)を挿入する。
これは，複数のテーブルを参照するので，
どのテーブルの列を指すのかはっきりさせるためである。
FROM 句に書くテーブル名は,(カンマ)で区切る。
WHERE 句の探索条件のところでも，列名の前にはテーブル名をつけて修飾する。
この結合処理の内部的な手順は，リレーショナル代数の表現でいうと，
直積，選択，射影をやっていることになる。
つまり，まずテーブル「社員通勤表」と「通勤手当早見表」との直積を求める。
つまり２つのテーブルの間のすべての行の組合わせを作成し，
それぞれの組合わせごとに列を横方向につなげて１つの行にする。
できた直積から，
探索条件「社員通勤表.住所 CODE = 通勤手当早見表.住所 CODE」を満たす行を選択する。
選択した行から，必要な列を抜き取る。
こうしたことをやっている。

例題 17 結合（join）：２つのテーブルを結合する(2)

```
SELECT B1.顧客，B1.売上商品，
       B1.数量，B2.単価円，
       B1.数量 * B2.単価円 AS 売上高
FROM   B1，B2
WHERE  B1.売上商品 = B2.商品;
```

２つのテーブル B1 と B2 とから，
「売上商品」と「商品」に記された商品名が等しい行を選択して，
列「顧客」，「売上商品」，「数量」，「単価円」を表示するとともに，
さらに「数量×単価円」で売上高を計算して
それに「売上高」という列名をつけて表示する。

例題 18 直積（Cartesian product）：２つのテーブルのペア

```
SELECT B1.*，B2.*
FROM B1，B2;
```

テーブル B1 とテーブル B2 とから，すべての行を対象に直積を作成し，
すべての列を表示する。
B1.*という表現は，テーブル B1 のすべての列という意味である。

例題 19 相関名によるテーブルの略記

```
SELECT X.社員名，
       Y.最寄り駅，
       Y.通勤手当円
FROM   社員通勤表 X，通勤手当早見表 Y
WHERE  X.住所 CODE = Y.住所 CODE;
```

テーブル名が長いと，それを列名の前につけて修飾するのが煩雑である。
そこで FROM 句で，「社員通勤表 X」（間にスペースを入れる）と書くと，
テーブル「社員通勤表」を単に X というテーブル名で参照することができるようになる。
これを相関名またはエイリアス(alias，別名)という。
後の FROM 句で相関名を定義しているのに，
先にある SELECT 句で先にそれを参照するのはプログラミングの順序からいうと
奇妙に感ずるかもしれない。
これは，SQL は手続きが書かれた順序で実行されるわけではなく，
SELECT...FROM...WHERE のまとまりとして意味を持っているからである。

例題 20 相関名による同一テーブルの参照

```
SELECT TAB1.数量－TAB2.数量
FROM   B1 TAB1, B1 TAB2
WHERE  TAB1.担当社員='橋本 龍子'
AND    TAB2.担当社員='亀井 勇';
```

テーブル B1 の「橋本 龍子」の数量と，
同じテーブルの「亀井 勇」の数量との差を計算して表示する。
相関名を使うと，同一テーブルに異なる相関名をつけて，
あたかも異なるテーブルであるかのように扱うことができる。
１つのテーブルの中の異なる行の間で比較したり計算したりするのに有効である。

例題 21 副問合せ(1)

```
SELECT 最寄り駅, 通勤手当円
FROM  通勤手当早見表
WHERE 通勤手当円>=
  (SELECT 通勤手当円
   FROM 通勤手当早見表
   WHERE 最寄り駅='調布'
  );
```

テーブル「通勤手当早見表」の中から，
最寄り駅が調布のときの通勤手当と比べて
それ以上の金額になる最寄り駅を表示する。
WHERE 句の探索条件で，述語（条件式）の値式（列名，定数，計算式）として
問合せ（SELECT...FROM...WHERE...）を使用することができる。これを副問合せという。
副問合せは，基本的に単一の列についての値を返すような問合せである。
返ってくる値は複数あってもよいが，
単一列についてのものであるという点がポイントである。
また複数の値が返るときは，述語（条件式）が IN 述語のように，
述語として整合のとれたものでなければならない。
上の例では，最寄り駅が調布のときの「通勤手当円」の値を１つだけ返している。
副問合せは，（ ）でくくる。
データ処理的には，主問合せでテーブルの１行ごとに
WHERE 句の探索条件が真かどうかの判定が行われる点はこれまでと同じである。
ただ探索条件の判定のステップで，そのつど副問合せが発生している。

例題 22 副問合せ(2)

```
SELECT 最寄り駅, 通勤手当円
FROM   通勤手当早見表
WHERE  住所 CODE IN
  (SELECT 住所 CODE
   FROM 社員通勤表
   WHERE 社員名 IN ('橋本 龍子','亀井 勇')
  );
```

「橋本 龍子」と「亀井 勇」の住所 CODE を

テーブル「社員通勤表」から副問合せで探索して，
探索した住所 CODE に該当する「最寄り駅」と「通勤手当円」を
テーブル「通勤手当早見表」から抜き出して表示する。
この副問合せでは，「橋本 龍子」と「亀井 勇」の住所 CODE が２つ返る。
IN 述語でそれを参照している。

例題 23 EXISTS 述語(1)

```
SELECT 社員名
FROM   A1
WHERE  EXISTS
  (SELECT *
   FROM   A2
   WHERE  A1.社員 CODE=A2.社員 CODE
  );
```

テーブル A1 と A2 の間で，社員 CODE が等しいものがあれば，
その社員名を表示する。A1 と A2 の「共通（intersection）」を求めている。
EXISTS 述語は，
    EXISTS (SELECT...FROM...WHERE...)
という形式をとり，
副問合せ SELECT...FROM...WHERE...で探索された要素が 1 つ以上あれば
（つまり空集合でなければ）真(true)の値をとる。空集合であれば，偽(false)となる。
この EXISTS 述語で使用する副問合せに限って，
SELECT 句で抜き出す列は単一でなくてよい。ふつうは
    SELECT * FROM ... WHERE ...
という形式をとる。

例題 24 EXISTS 述語(2)：NOT EXISTS

```
SELECT *
FROM A1
WHERE NOT EXISTS
  (SELECT *
   FROM   A2
   WHERE  A1.社員 CODE=A2.社員 CODE
  );
```

テーブル A1 にあって A2 に存在しない社員の情報を抜き出している。
つまり，A1－A2（difference，差）を求めている。
NOT EXISTS 述語は，
    NOT EXISTS (SELECT...FROM...WHERE...)
という形式をとり，
副問合せ SELECT...FROM...WHERE...で探索された要素が 1 つもなければ
（つまり空集合であれば）真(true)の値をとる。空集合でなければ，偽(false)となる。

例題 25 UNION

```
(SELECT * FROM A1)
UNION
(SELECT * FROM A2);
```

テーブル A1 とテーブル A2 の和（union）を求めている。重複する行（すべての列の値が等しい行どうし）は除かれて１つの行だけが選択される。

例題 26 ALL 述語

```
SELECT B1.担当社員, B1.数量*B2.単価円 AS 売上高
FROM   B1, B2
WHERE  B1.売上商品=B2.商品
AND    売上高 >= ALL
  (SELECT X.数量*Y.単価円
   FROM   B1 X, B2 Y
   WHERE  X.売上商品=Y.商品
  );
```

他のすべての担当社員の売上高以上の売上高をもつ担当社員の情報を表示する。
つまり最大の売上高をあげた社員を選択している。
ALL 述語は，
　a 比較演算子 ALL(SELECT b FROM c WHERE d)
という形式をとる。a は値式，b は列，c はテーブル，d は探索条件である。
副問合せで返ってくるすべての値に対して比較演算が真(true)のとき，
ALL 述語は真(true)の値をとる。
また上の形式の ALL 述語は，EXISTS 述語を用いて次のように書きかえることができる。
　NOT EXISTS (SELECT * FROM c WHERE (d) AND NOT(a 比較演算子 c.b)

例題 27 ANY 述語(SOME 述語，ANY 述語)

```
SELECT B1.担当社員, B1.数量*B2.単価円 AS 売上高
FROM   B1, B2
WHERE  B1.売上商品=B2.商品
AND    売上高 > SOME
  (SELECT X.数量*Y.単価円
   FROM   B1 X, B2 Y
   WHERE  X.売上商品=Y.商品
  );
```

少なくとも他の１人の社員よりは大きな売上高をあげた社員を選択する。
つまり最小の売上高ではない社員を選択する
SOME 述語（ANY を用いても同じ）は，
　a 比較演算子 SOME(SELECT b FROM c WHERE d)
という形式をとる。a は値式，b は列，c はテーブル，d は探索条件である。
副問合せで返ってくる値のうち少なくとも１つに対して比較演算が真(true)のとき，
SOME 述語は真(true)の値をとる。
また上の形式の SOME 述語は，EXISTS 述語を用いて次のように書きかえることができる。
　EXISTS (SELECT * FROM c WHERE (d) AND (a 比較演算子 c.b)

例題 28 商（division）

```
SELECT DISTINCT スーパー
FROM C1
WHERE NOT EXISTS
( SELECT DISTINCT X1.スーパー, X2.商品種類,X2.メーカー
  FROM C1 X1, C2 X2
  WHERE NOT EXISTS
  ( SELECT *
    FROM C1 Z
    WHERE  X1.スーパー=Z.スーパー
    AND    X2.商品種類=Z.商品種類
    AND    X2.メーカー=Z.メーカー
  )
  AND   C1.スーパー=X1.スーパー
);
```

商 C1÷C2 を求めている。
商は，
Q＝C1[スーパー]－（C1[スーパー]×C2－C1）[スーパー]
で求められる。
なおここで，[スーパー]とは「スーパー」の列を取出す射影である。また，×は直積で，
－は差である。
もっとスマートな表現があるかもしれない。

以上