

目次

■ 変数のスコープと寿命	1
■ プロシージャのスコープ定義	1
■ 列変数のスコープ	2
■ ローカル変数は静的変数	3
■ プロシージャ引数の引渡し方法（値渡しと参照渡し）	4
■ 変数の定義	4
■ 変数の種類と型	6
■ 変数の命名	10
■ Load-Lookup, Lookup コマンドによる最適化	12

■変数のスコープと寿命

変数にはスコープ（有効範囲）と寿命（生存期間）があります。スコープは変数へのアクセスが可能な範囲のことです。スコープには、グローバル変数がアクセス可能なグローバルスコープと、ローカル変数が使えるプロシージャ個々のローカルスコープがあります。寿命は変数が生存している（値を保持する）実行上の期間です。

SQR のローカル変数の寿命は一般のプログラミング言語と違います。SQR のローカル変数は静的変数です。静的変数はプロシージャ実行後も値を保持します。

■プロシージャのスコープ定義

プロシージャの定義には以下の形式があります。定義の仕方によってローカル変数が使えるようになります。

グローバル・スコープのみ持つプロシージャの定義

```
Begin-Procedure ProcName
```

パラメータもなく Local 指定子もないプロシージャはローカルスコープを持ちません。この形式のプロシージャ内ではローカル変数は使えません。変数は全てグローバル変数となります。

ローカル・スコープを持つプロシージャの定義

```
Begin-Procedure ProcName Local  
Begin-Procedure ProcName ()  
Begin-Procedure ProcName (#a, ... #z)
```

Local が指定されたプロシージャ、または、0 個以上のパラメータが定義されたプロシージャは、ローカルスコープを持ちます。この形式のプロシージャ内ではローカル変数が使えます。**この形式のプロシージャ内でグローバル変数を使うには変数名の前にアンダースコアを付ける必要があります。**

例 1. グローバル・スコープのみ持つプロシージャ

```
Begin-Procedure Proc1
  Let $foo = 'abc'           ! "foo" という名前のグローバル変数.
  Let $_foo = '123'         ! "_foo" という名前のグローバル変数.
End-Procedure
```

例 2. ローカル・スコープを持つプロシージャ

```
Begin-Procedure Proc2 Local
  Let #a = 123               ! ローカル変数.
  Let $_foo = 'def'         ! "foo" という名前のグローバル変数.
  Let $__foo = 'efg'        ! "_foo" という名前のグローバル変数.
End-Procedure

Begin-Procedure Proc3(#x, #y)
  Let #a = 456               ! ローカル変数. Proc2 の #a とは別な変数
End-Procedure
```

■列変数のスコープ

ローカル・スコープ内でグローバルな列変数（&で始まる変数）を使う場合は、通常の変数と同様に変数名の前にアンダースコアを付けます。

同一のスコープ内で同じ名前の列変数を使うことは出来ません。例 1 では列変数 &EMPLID がグローバル・スコープの異なる Begin-Select で使われているのでコンパイル・エラーになります。

逆にスコープが異なる列変数には同じ名前が使えます。

例 1. 同じスコープ内で同じ名前の列変数を使用するとエラーになる

```
Begin-Procedure foo           ! foo のスコープはグローバル.

Begin-Select
  EMPLID      &EMPLID
FROM PS_PERSON
End-Select

End-Procedure

Begin-Procedure bar           ! bar のスコープはグローバル.

Begin-Select
  EMPLID      &EMPLID         ! foo で既に使用されているのでエラー.
FROM PS_PERSON
End-Select

End-Procedure
```

例 2. スコープが異なる列変数には同じ名前が使える

```
Begin-Procedure foo                ! foo のスコープはグローバル.

Begin-Select
EMPLID      &EMPLID
FROM PS_PERSON
End-Select

End-Procedure

Begin-Procedure bar Local          ! bar のスコープはローカル.

Begin-Select
EMPLID      &EMPLID                ! foo の &EMPLID と名前が重複しても
                                   ! スコープが異なるので問題ない
FROM PS_PERSON
End-Select

    Show &_EMPLID                  ! foo で定義されたグローバル列変数 &EMPLID を参照.

End-Procedure
```

■ ローカル変数は静的変数

ローカル変数は静的変数です。つまり、プロシージャの呼び出し後も変数の値は保持されます。このため、呼び出し後に値を保持する必要がない場合、**SQR ではプロシージャのローカル変数は明示的に初期化すべき**です。プロシージャのパラメータもローカル変数であり静的です。下記のコードはローカル変数が静的であることを示します。foo の2回目の呼び出しでは 2 が印刷されます。

例. ローカル変数は呼出し後も値を保持する

```
Begin-Program
    Do foo        ! 1 が印刷される
    Do foo        ! 2 が印刷される
End-Program

Begin-Procedure foo Local
    Declare-Variable
        Integer #i
    End-Declare
    Let #i = #i + 1
    Show #i
End-Procedure
```

自動変数が存在しない SQR では再帰関数が書けません¹。SQR プログラムで再帰関数が必要になることは稀ですが、もし、木構造などのオブジェクトを再帰処理する必要がある場合には、一時テーブルや配列などを利用することになるでしょう。

参考までに PeopleCode について言えば、変数を関数内部で Local number &i; のように明示的に定義した場合は自動変数となります。関数外部で定義される変数および明示的に定義しない変数は静的変数となります。

¹ 末尾再帰などスタックを利用しない再帰関数なら書けます。

■プロシージャ引数の引渡し方法（値渡しと参照渡し）

S Q Rでは参照渡しを使ってプロシージャの結果を返します。プロシージャ定義でパラメータの前に：（コロン）を付加すれば、そのパラメータを参照渡しにすることができます。

例. 値渡し

値渡しは、引数を呼び出し側で設定しても、呼び出し元の変数には影響を与えません。

```
Begin-Program
  Let #a = 0
  Do foo(#a)
  Show #a    ! 0 が印刷される
End-Program

Begin-Procedure foo(#a)
  Let #a = 5
End-Procedure
```

例. 参照渡し

参照渡しは、呼び出し側での引数の設定が、呼び出し元の変数に反映されます。

```
Begin-Program
  Let #a = 0
  Do bar(#a)
  Show #a    ! 5 が印刷される
End-Program

Begin-Procedure bar(:#a)    ! パラメータの前にコロンを付加.
  Let #a = 5
End-Procedure
```

■変数の定義

変数の定義については以下のとおりです。

- Declare-Variable コマンドを使用することで、変数を明示的に定義できます。
- 明示的に定義されていない変数は、使用時に自動で定義されます。
- グローバル変数を定義するには、Begin-Setup ブロック内で Declare-Variable コマンドを使用します。1 プログラム内に複数の Begin-Setup ブロックを記述してもエラーにはなりませんが²、Declare-Variable による変数の定義は、ファイル上、その変数が使われるより前に記述されていなければなりません。
- ローカル変数およびプロシージャのパラメータを明示的に定義するには、ローカルとして定義されたプロシージャの最初で Declare-Variable コマンドを使用します。

² 何故か Begin-Program も1プログラム内に何個書いてもエラーにはなりません。実行されるのは最初のものだけです。

例. Declare-Variable の使用例

※変数の型については、「変数の種類と型」の節を参照して下さい。

```
Declare-Variable
  Default-Numeric=Integer
  Date      $dt
  Text      $Emplid $Message
  Integer    #i #j
  Float      #x
             #y
  Decimal(5) #dec
End-Declare
```

例. Begin-Setup セクションにおけるグローバル変数の定義

```
1: Begin-Setup
2:   Declare-Variable
3:     integer #a
4:   End-Declare
5: End-Setup
6:
7: Begin-Program
8:
9:   Let #a = 3
10:  Let #b = 3
11:
12: End-Program
13:
14: Begin-Setup
15:   Declare-Variable
16:     integer #b      ! コンパイル・エラー.  #b は 10 行目で自動的に定義済み.
17:     integer #c      ! OK
18:   End-Declare
19: End-Setup
20:
21: Begin-Procedure foo
22:
23:   Show #c
24:
25: End-Procedure
```

プロシージャのパラメータもローカル変数と同様に定義できます。特に、パラメータを日付変数として使用する場合には、明示的な定義が必要になります。

例. プロシージャのパラメータを明示的に定義する場合

```
Begin-Program
  Do foo('20130501')
End-Program

Begin-Procedure foo($dt)
  Declare-Variable
    Date $dt      ! $dt は DATE 変数 である
  End-Declare

  Let $dt = dateadd($dt, 'month', 1)  ! コンパイルOK
  Show $dt
End-Procedure
```

例. プロシージャのパラメータを明示的に定義しない場合

```
Begin-Program
  Do foo('20130501')
End-Program

Begin-Procedure foo($dt)

  Let $dt = dateadd($dt, 'month', 1)  ! コンパイル・エラー
                                     ! $dt は デフォルトで TEXT 変数となるため.
  Show $dt

End-Procedure
```

■ 変数の種類と型

変数には数値変数、文字列変数、列変数の3つの種類があります。それぞれ、#、\$、& で始まる変数です。変数の各種類には、更なる詳細な型付けがあります。

数値変数の型付け

数値変数は以下のいずれかの型を持ちます。

INTEGER	整数型
FLOAT	2 進の浮動小数点数
DECIMAL(n)	有効桁 n 桁の 10 進浮動小数点数

10 進の浮動小数点数

小数は基数（10 進数なら基数は 10）によって有限になったり無限になったりします。たとえば、10 進数の有限小数 0.1 を 2 進数で表現すると 0.00011001100110011... と無限小数になります。10 進数の 0.1 は、2 進の浮動小数点数で表現すると近似値に丸められます。一方、10 進の浮動小数点数を使うと、有効桁以内の 10 進数の有限小数であれば、正確に表現することができます。

SQL の DECIMAL 型は、SQL92 の DECIMAL 型とは違い固定小数点数ではありません。しかし、整数桁 n 、小数桁 s の固定小数点数の全ての数値は、有効桁 n + s の 10 進の浮動小数点数で正確に表現できます。

例. DECIMAL 型の使用例

```
Begin-Setup
  Declare-Variable
    Decimal(5) #a
  End-Declare
End-setup

Begin-Program

  Let #a = 1234567890
  Show #a

  Let #a = 12345.67890
  Show #a

  Let #a = 123.4567890
  Show #a
```

```
End-Program
```

実行結果

```
1234500000.000000  
12345.000000  
123.450000
```

デフォルトの数値型

明示的な定義がない数値変数は、デフォルトの数値型で定義されます。このデフォルトの数値型は以下の方法で、INTEGER、FLOAT、DECIMAL の型のいずれかを指定します。より上のものが優先されます。

- ① Declare-Variable 内で Default-Numeric で定義する。
- ② コマンドライン引数 -DNT を使って指定する。
- ③ SQR.ini の [Default-Settings] セクションで DEFAULT-NUMERIC を使って指定する。
- ④ 上記のいずれにも該当しない場合は FLOAT となる。

文字列変数の型付け

文字列変数は、以下のいずれかの型を持ちます。

DATE	日付
TEXT	テキスト

DATE 型が TEXT 型と異なる点は以下のとおりです。

- DATE 型の変数を使用するには、Declare-Variable による明示的な定義が必要です。
- DATE 型の変数に対しては、以下の組み込みの日付関数が使えます。TEXT 型の変数に対して、以下の関数を使うと、コンパイル・エラーとなります。

```
dateadd  
datediff  
datenow  
datetostr
```

- 大小比較を行う条件式では、両辺のどちらか一方が DATE 型の値である場合には日付順で比較されます。このとき、他方が日付値に変換できない場合、実行時エラーが発生します。通常の文字列の比較、すなわち、辞書順による比較は、両辺のどちらも DATE 型の値でない場合のみ行われます。

例. 日付値の比較

```
Begin-Setup
  Declare-Variable
    Date $date1 $date2 $date3
    Text $text1 $text2
  End-Declare
End-setup

Begin-Program
  Let $text1 = '30-MAR-13'
  Let $text2 = '30-APR-13'

  Let $date1 = $text1
  Let $date2 = $text2

  If $date1 < $date2                ! 日付順で比較
    Show '$date1 < $date2'
    Show $date1 noline
    Show ' < ' noline
    Show $date2
  End-If

  If $text1 > $text2                ! 辞書順で比較
    Show '$text1 > $text2'
    Show $text1 noline
    Show ' > ' noline
    Show $text2
  End-If

  ! どちらか一方が日付値ならば、日付順で比較
  If $text1 < dateadd($date2, 'day', -7)
    Show '$text1 < dateadd($date2, ''day'', -7)'
  End-If
End-Program
```

実行結果

```
$date1 < $date2
30-MAR-13 < 30-APR-13
$text1 > $text2
30-MAR-13 > 30-APR-13
$text1 < dateadd($date2, 'day', -7)
```

文字列から日付値への変換

最初に結論を言うと、SQLR には日付リテラルの確実な記法がありません。日付定数を設定する場合、次のように明示的に変換書式を指定するのが無難です。

```
Let $dt = strtodate('2012-12-31', 'YYYY-MM-DD')
```

以降では、明示的に変換書式を指定しない場合、たとえば、次のように文字列を直接、DATE 型の変数に設定する場合の変換仕様について説明します。

```
Let $dt = '2012-12-31'
```

変換書式が指定されない文脈で、文字列値が日付変数に設定された場合、以下のステップで文字列から日付値への変換が試みられます。

1. SQR_DB_DATE_FORMAT の書式で変換する。
2. デフォルトのデータベース依存書式で変換する。
3. データベース非依存の書式で変換する。

SQR_DB_DATE_FORMAT は `sqr.ini` で定義した書式です。デフォルトのデータベース依存書式³は組み込みの書式であり、データベースが Oracle の場合なら、'DD-MON-YY' です。3 番目のデータベース非依存の書式は 'SYYYMMDD[HH24[MI[SS[NNNNNN]]]]' です。この書式の先頭にある S (Sign) は、オプションであり、'-' (マイナス) を指定すれば BC 年となります。

DATE 変数に、いずれの書式にも合致しない文字列を設定しようとした場合、実行時エラーが起こります。その際、下記のようなメッセージが表示されます。

```
Error on line 9:
(SQR 1944) The date '2013/04/24' is not in the format
specified by SQR_DB_DATE_FORMAT or in one of the accepted
formats listed below:
      DD-MON-YY
      SYYYMMDD[HH24[MI[SS[NNNNNN]]]]
```

データベース非依存の書式

変換ステップの 3 番目にあったデータベース非依存の書式の使用には注意が必要です。変換ステップの優先度の関係上、データベース非依存の書式を使用したコードの実行結果が `sqr.ini` の設定によって、変わることがあります。

例えば、次のコードの代入命令 (1 番目の Let 文) が、下記、`sqr1.ini` と `sqr2.ini` の場合とでは異なる動作をします。前者の場合には、データベース非依存の書式が適用され、後者の場合、SQR_DB_DATE_FORMAT の書式が適用されます。したがって、データベース非依存の書式を使わず、明示的に変換書式を指定するのが無難でしょう。

³ デフォルトのデータベース依存書式については、SQR のコマンド・リファレンスの PRINT コマンドを参照して下さい。

例. 文字列から日付への変換

```
Begin-Setup
  Declare-Variable
    date $dt
  End-Declare
End-setup

Begin-Program

  Let $dt = '20121231'
  Show $dt edit 'AD.YYYY-MM-DD'

  Let $dt = '-20121231'
  Show $dt edit 'AD.YYYY-MM-DD'

End-Program
```

sqr1.ini の設定

```
;sqr1.ini
[Environment:Oracle]
SQR_DB_DATE_FORMAT=DD-MON-YYYY
```

sqr1.ini での実行結果

```
AD.2012-12-31
BC.2012-12-31
```

sqr2.ini の設定

```
;sqr2.ini
[Environment:Oracle]
SQR_DB_DATE_FORMAT=DDMMYYYY
```

sqr2.ini での実行結果

```
AD.1231-12-20
BC.2012-12-31
```

■変数の命名

SQR は構文仕様が不完全なため、変数名に使う文字は以下の文字だけに限定することを勧めます。以下のリストには英字の大文字・小文字がありますが、変数名の大文字・小文字は区別されません。

```
0～9, A～Z, a～z, '_', '-'
```

#A-3 といった変数名は避けるべきです。このような名前の変数を Let 文の右辺や IF 文の条件式で使用する警告が出ます。

以降は変数名の不具合について説明します。

他のプログラミング言語では考えられませんが、SQR では文脈によって変数名に使用可能な文字が違います。例えば Declare-Variable では変数名に使用可能な文字には制限がありますが、自動定義される変数名には、ほとんど制限がありません。

例を示します。

```
Begin-Program
  Move 3 to #AB, (CD)+
  Show #AB, (CD)+
End-Program
```

このコードは問題なく動きます。Move や Show などの古くからあるコマンドは、ほとんど任意の変数名が使えます。AB, (CD)+ は、有効な変数名です。しかし、この変数名は、Declare-Variable や Let 文、Do 文では使えません。これを Declare-Variable で使うと、変数名が不正でコンパイル・エラーになります。Let 文、Do 文で使えば構文解析でエラーになります。

もうひとつ例を挙げます。Let 文で有効な変数名でも Declare-Variable では無効なものがあります。Let 文だけの次のプログラムは問題なく動作します。

```
Begin-Program
  Let #AB.C = 4
  Show #AB.C
End-Program
```

この変数を、以下のように Declare-Variable で定義するとコンパイル・エラーが発生します。

```
Begin-Setup
  Declare-Variable
    Integer #AB.C
  End-Variable
End-Setup

Begin-Program
  Let #AB.C = 4
  Show #AB.C
End-Program
```

実際に出力されるエラーは次のようなものです。

```
Error on line 3:
(SQR 7210) Invalid variable name specified.
Integer #AB.C
```

■ Load-Lookup, Lookup コマンドによる最適化

例として下記の手順で従業員の住所一覧を作成するものとします。

1. 住所テーブル ADDRESSES を Begin-Select し、取得した各行に対して次の 2、3 を実行する。
2. 現在行の都道府県コードをもとに、STATE_TBL⁴ から、都道府県名を取得する。
3. 2 で取得した都道府県名とその他の住所フィールドを連結し、明細行に出力する。

2 の都道府県名を取得する方法ですが、単純に考えられるのは都度、Begin-Select で STATE_TBL を問合せることです。しかし、この方法は対象者の人数分、SQL の発行回数が増えるのが難点です。

STATE_TBL の日本のレコードは 47 行しかありません。Load-Lookup と Lookup コマンドを使えば、この 47 行を一旦バッファに読み込み、このバッファから県名を引用できます。

サンプル・コードを以下に示します。

例. Load-Lookup と Lookup コマンドの使用例

Begin-Program

! LOAD-LOOKUP コマンドでデータをロードする

LOAD-LOOKUP

NAME = Prefectures
TABLE = PS_STATE_TBL
KEY = STATE
RETURN_VALUE = DESCR
WHERE = COUNTRY='JPN'
SORT = SI
QUIET

! バッファの名称 (任意)

! 取得先テーブル名

! KEY となるフィールド名

! 値を持つフィールド名

! 取得先の抽出条件

! 大文字・小文字区別なしにソートする

! 余計なメッセージ出力を抑制する

! LOOKUP コマンドで値を表引きする

LOOKUP Prefectures 13 \$Descr

! 都道府県コード 13 の名称を取得する

Show \$Descr

End-Program

⁴ 正確を期すれば \$PSOptions_Language_Cd (prcslng.sqc) が 'JPN' ではない場合は、STATE_NAMES_LNG を参照すべきです。
ここでは簡略化のためそこまで考慮していません。

上記のコードで LOAD-LOOKUP が実行されたとき、次のような SQL が発行されます。

```
SELECT DISTINCT
    STATE.....KEY で指定したフィールド名
,    DESCR.....RETURN_VALUE で指定したフィールド名
FROM
    PS_STATE_TBL ..... TABLE で指定したテーブル名
WHERE
    COUNTRY='JPN' .....WHERE で指定した抽出条件
```

標準の SQR には、トランスレート値を LOAD-LOOKUP で読み込んでいるものがあります⁵。トランスレート値は有効日管理されているので、最新のトランスレート値のみ出力するという条件付きで、LOAD-LOOKUP を使用することができます。

例. LOAD-LOOKUP でトランスレート値を取得する⁶

Begin-Program

```
Let $Criteria = 'FIELDNAME='GPJP_IT_TYPE'
                AND EFFDT = (SELECT MAX(EFFDT)
                             FROM PSXLATITEM B
                             WHERE B.FIELDNAME = A.FIELDNAME
                             AND B.FIELDVALUE = A.FIELDVALUE
                             AND B.EFF_STATUS = 'A')'
```

LOAD-LOOKUP

```
NAME           = XlatGpjpItType
TABLE          = 'PSXLATITEM A'
KEY            = FIELDVALUE
RETURN_VALUE   = XLATLONGNAME
WHERE          = $Criteria
SORT           = SI
QUIET
```

LOOKUP XlatGpjpItType '10' \$LongName

Show \$LongName

End-Program

⁵ 1dpIntyp.sqc がよい参考になります。

⁶ ここでもやはり \$PSOptions_Language_Cd (prcsIng.sqc)が 'JPN' ではない場合は、PSXLATITEMLANG を参照すべきですが、簡略化のためそこまで考慮していません。