一　数据类型和对象

在 ABAP中，可以使用与标准数据声明相似的语法处理数据类型，而与数据对象无关。

在程序中必须声明要使用的全部数据对象。　　声明过程中，　必须给数据对象分配属性，　　其中最重要的属性就是数据类型。

1.1　基本数据类型

数据类型	默认大小	有效大小	初始值	说明
C	1	1 - 65535	SPACE	文本、字符（字母数字字符）
D	8	8	'00000000'	日期（格式：　YYYYMMDD）
F	8	8	0	浮点数
I	4	4	0	整型（整数）
N	1	1 - 65535	'00...0'	数字文本
P	8	1 - 16	0	压缩号
T	6	6	'000000'	时间（格式：　HHMMSS）
X	1	1 - 65535	X'00'	十六进制

对算术运算的非整型结果（如分数）进行四舍五入，而不是截断。

类型 P 数据允许在小数点后有数字。　有效大小可以是从　1　到 16　字节的任何值。 将两个十进制数字压缩到一个字节，　而最后一个字节包含一个数字和符号。　在小数点后最多允许 14 个数字。

1.2　系统定义的数据对象

SPACE	空白字符串
SY-SUBRC	系统执行传回值，　0 表示成功
SY-UNAME	用户
SY-DATUM	系统日期
SY-UZEIT	系统时间
SY-TCODE	当前执行程序的　Transaction code
SY-INDEX	当前 LOOP循环过的次数
SY-TABIX	当前处理的是　Internal Table(　内部表) 的第几行
SY-TMAXL	Internal Table　（内部表）的总行数
SY-SROWS	屏幕总行数
SY-SCOLS	屏幕总列数
SY-VLINE	画竖线
SY-ULINE	画横线
SY-PAGNO	当前页号
SY-LINSZ	当前报表宽度
SY-LINCT	当前报表长度
SY-LISND	列表索引页
SY-LISTI	上一个列表的索引
SY-LILLI	绝对列表中选定的行号
SY-CUROW	屏幕上的行

	光标列
	真实行号
	选择行的内容，长度为 255
	当前行

1.3 确定数据对象的属性

如果要查明数据对象的数据类型，或者要在程序的运行期间使用其属性，可使用DESCRIBE语句。语法如下：

DESCRIBE FIELD <f> [LENGTH <l>] [TYPE <t> [COMPONENTS <n>]]

[OUTPUT-LENGTH <o>] [DECIMALS <d>]

[EDIT MASK <m>].

将由语句的参数指定的数据对象 <f> 的属性写入参数后的变量。

DESCRIBE FIELDS语句具有下列参数：

参数	用途
LENGTH	确定字段长度
TYPE	确定数据类型
OUTPUT-LENGTH	确定输出长度
DECIMALS	确定小数位
EDIT MASK	确定转换例程

1.3.1 确定字段长度

要确定数据对象的长度，利用 DESCRIBE FIELD 语句使用 LENGTH 参数，如下所示：

DESCRIBE FIELD <f> LENGTH <l>.

系统读取字段 <f> 的长度，并将值写入字段 <l> 。

1.3.2 确定数据类型

要确定字段的数据类型，利用 DESCRIBE FIELD 语句使用 TYPE 参数，如下所示：

DESCRIBE FIELD <f> TYPE <t> [COMPONENTS <n>].

系统读取字段 <f> 的数据类型，然后将值写入字段 <t> 。

除返回预定义数据类型 C、D、F、I、N、P、T 和 X 外，该语句还返回

2 s 对于带前导符号的两字节整型

2 b 对于无前导符号的一字节整型

2 h 对于内表

2 C 对于组件中没有嵌套结构的结构

2 C 对于组件中至少有一个嵌套结构的结构

1.3.3 确定输出长度

要确定字段的输出长度，利用 DESCRIBE FIELD 语句使用 OUTPUT-LENGTH参数，如下

所示：

DESCRIBE FIELD <f> OUTPUT-LENGTH <o>.

系统读取字段 <f> 的输出长度，并将值写入字段 <o>。

1.3.4 确定小数位

若要确定类型 P 字段的小数位的个数， 利用 DESCRIBE FIELD 语句使用 DECIMALS参数，如下所示：

DESCRIBE FIELD <f> DECIMALS <d>.

系统读取字段 <f> 的小数个数，并将值写入字段 <d>。

1.3.5 确定转换例程

要确定 ABAP/4 词典中某字段的转换例程是否存在，如果存在，名称是什么，利用 DESCRIBE FIELD 语句使用 EDIT MASK 参数，如下所示：

DESCRIBE FIELD <f> EDIT MASK <m>.

如果 ABAP/4 词典中字段 <f> 的转换例程存在， 则系统将其写入字段 <m>，然后将系统字段 SY-SUBRC 中的返回代码设为 0 。

可以像下面所显示的那样，在 WRITE 语句中将字段 <m>直接用作格式模板：

WRITE <f> USING EDIT MASK <m>.

如果字段 <f> 没有转换例程，则系统将返回代码设为 4 。

二数据输出到屏幕

2.1 在屏幕上定位 Write 输出

通过制定字段名称前面的格式规范，可以在屏幕上定位 WRITE 语句的输出：

WRITE AT [/][<pos>][(<len>)] <f>.

此处

3 斜线‘ / ’表示新的一行

3 <pos>是最长为三位数字的数字或变量，表示在屏幕上的位置

3 <len> 是最长为三位数字的数字或变量，表示输出长度

如果格式规范只包含直接值（即，不是变量） ，可以忽略关键字 AT。

2.2 格式化选项

对 WRITE 语句，可以使用不同的格式化选项。

WRITE ....<f>< 选项 >.

（1）所有数据类型的格式化选项：

选项	用途
LEFT-JUSTIFIED	输出左对齐
CENTERED	输出居中
RIGHT-JUSTIFIED	输出右对齐
UNDER <g>	输出直接开始于字段 <g>下
NO-GAP	忽略字段 <f> 后的空格
USING EDIT MASK <m>	指定格式模板 <m>
USING NO EDIT MASK	撤消对 ABAP/4 词典中指定的格式模板的激活。
NO-ZERO	如果字段仅包含零，则用空格代替它们 对类型 C 和 N 字段，将自动代替前导零

（2）数字字段的格式化选项：

选项	用途
NO-SIGN	不输出前导符号
DECIMALS <d>	<d>定义小数点后的数字位数。
EXPONENT <e>	在类型 F 字段中，在 <e>中定义幂数
ROUND <r>	用 10**(-r) 乘类型 P 字段，然后取整
CURRENCY <c>	按表格 TCURX 中的货币 <c>格式化。
UNIT <u>	按表格 T006 中为类型 P 字段所指定的单位 <u>固定小数位数

（3）日期字段的格式化选项：

选项	用途
DD/MM/YY	用户主记录中定义的分隔符
MM/DD/YY	用户主记录中定义的分隔符
DD/MM/YYYY	用户主记录中定义的分隔符
MM/DD/YYYY	用户主记录中定义的分隔符
DDMMYY	无分隔符
MMDDYY	无分隔符
YYMMDD	无分隔符

2.3 输出符号和图标

使用下列语法，可以在屏幕上输出符号和 R/3 图标：

WRITE <symbol-name> AS SYMBOL.

WRITE <icon-name> AS ICON.

符号和图标的名称（ <符号名 >和<图标名 >）是定义系统的常量，这些常量在包含程序 <SYMBOL>和<ICON>（尖括号是名称的一部分） 中指定。 这些包含程序也包含符号和图标的简短说明。输出符号和图标最简单的方法是使用语句结构。

要使符号和图标对程序可用， 必须在程序中输入恰当的包含程序或更易理解的包含程序 <LIST>。

2.4 输出线和空行

（1）水平线

用下列语法，可以在输出屏幕上生成水平线：

ULINE [AT [/][<pos>][(<len>)]].

它等同于

WRITE [AT [/][<pos>][(<len>)]] SY-ULINE.

AT 后的格式规范，与在屏幕上定位 WRITE 输出中为 WRITE 语句说明的格式规范完全一样。如果没有格式规范，系统则开始新的一行，并用水平线填充该行。否则，只按指定输出水平线。

生成水平线的另一种方法，是在 WRITE 语句中键入恰当数量的连字符，如下所示：

WRITE [AT [/][<pos>][(<len>)]] '-----...'.

（2）垂直线

用下列语法，可以在输出屏幕上生成垂直线：

WRITE [AT [/][<pos>]] SY-VLINE.

或

WRITE [AT [/][<pos>]] '|'.

（3）空行

用下列语法，可以在输出屏幕上生成空行：

SKIP [<n>].

该语句从当前行开始， 在输出屏幕上生成 <n>个空行。 如果没有指定 <n>的值， 就输出一个空行。

（4）要将输出定位在屏幕的指定行上，使用：

SKIP TO LINE <n>.

该语句允许将输出位置向上或向下移动。

2.5 将字段内容作为复选框输出

使用下列语法，可以将字段的第一个字符，作为复选框输出到输出屏幕上：

WRITE <f> AS CHECKBOX.

如果字段 <f> 的第一个字符是一个“ X ”，就显示复选框已填充。如果字段 <f> 的第一个字符是 SPACE，就显示复选框为空。

该语句创建的复选框， 默认状态是可输入的。 就是说， 用户可以通过单击鼠标来填充它们或使其为空。

三赋值

在 ABAP/4 中，可以在声明语句和操作语句中给数据对象赋值。在声明语句中，将初始值赋给声明的数据对象。为此，可以在 DATA、常量或 STATICS 语句中使用 VALUE 参数。

要在操作语句中给数据对象赋值，可以使用：

6 MOVE语句和 WRITE TO 语句，对应于赋值运算符（=）

3.1 用 MOVE赋值

3.1.1 基本赋值操作

要将值（文字）或源字段的内容赋给目标字段，可以使用 MOVE语句或赋值运算符 （=）。

MOVE语句的语法如下所示：

MOVE <f1> TO <f2>.

MOVE语句将源字段 <f1> 的内容传送给目标字段 <f2> 。<f1> 可以是任何数据对象。<f2> 必须是变量，不能是文字或常量。传送后，<f1> 的内容保持不变。

赋值运算符（=）的语法如下所示：

<f2> = <f1>.

MOVE语句和赋值运算符功能相同。

3.1.2 用指定偏移量赋值

可以在每条 ABAP/4 语句中为基本数据类型指定偏移量和长度。在这种情况下，MOVE 语句语法如下：

MOVE <f1>[+<o1>][(<l1>)] TO <f2>[+<o2>][(<l2>)].

将字段 <f1> 从 <o1>+1 位置开始且长度为 <l1> 的段内容赋给字段 <f2> , 覆盖从 <o2>+1 位置开始且长度为 <l2> 的段。

在 MOVE 语句中，所有偏移量和长度指定都可为变量。

3.1.3 在字符串组件之间赋值

描述的 MOVE 语句赋值方法适用于基本数据对象和结构化数据对象。另外，还有一种 MOVE语句变体，允许将源字段串组件内容复制到目标字段串组件中。语法如下：

MOVE-CORRESPONDING <string1> TO <string2>.

该语句将字段串 <string1> 组件的内容赋给有相同名称的字段串 <string2> 组件。

对于每对名称，系统都执行 MOVE 语句，如下所示：

MOVE STRING1-<component> TO STRING2-<component>.

系统分别执行所有必要类型转换。该处理不同于包括整个字段串的赋值。在这种情况下，应用不兼容的字段串和基本字段所述的转换规则。

3.2 用 WRITE TO赋值

要将值（文字）或源字段内容写入目标字段，可以使用 WRITE TO 语句：

WRITE <f1> TO <f2> [<option>].

WRITE TO 语句将源字段 <f1> 内容写入目标字段 <f2> 。 <f1> 可以是任何数据对象， <f2> 必须是变量，不能是文字或常量。写入后， <f1> 内容保持不变。

对于 <option> ，可以使用 WRITE 语句的所有格式化选项（ UNDER和 NO-GAP 除外）。

WRITE TO 语句总是检查用户主记录中的设置。例如，这些设置指定是将小数点显示为句号（ .），还是逗号（ ,）。

WRITETO 语句并不遵循类型转换中所述的转换规则。目标字段解释为类型 C 字段。系统总是将源字段内容转换为类型 C，它不将结果字符串转换为目标字段的数据类型， 而直接写入目标字段。因此，不应使用数值数据类型的目标字段。

3.2.1 运行时指定源字段

可以使用 WRITE TO 语句在运行时指定源段。为此，需用括号将包含源字段名的数据对象名括起来，并将其放在源字段位置：

WRITE (<f>) TO <g>.

系统将赋给 <f> 的数据对象值放到 <g>中。

然而，如果要在运行时指定目标字段，则必须按使用字段符号中所述，使用字段符号。

3.2.2 用指定偏移量写入值

可以指定每条 ABAP/4 语句中基本数据对象的偏移量和长度。 WRITETO 语句语法如下：

WRITE <f1>[+<o1>][(<l1>)] TO <f2>[+<o2>][(<l2>)].

将字段 <f1> 中从 <o1>+1 位置开始且长度为 <l1> 的内容赋给字段 <f2> , 覆盖从 <o2>+1 位置开始且长度为 <l2> 的段。

在 WRITE TO 语句中，目标字段的偏移量和长度指定可为变量。

四数值运算

要处理数值数据对象并将结果值赋给数据对象，可以用 COMPUTE语句或赋值运算符（=）。 COMPUTE语句语法如下所示：

COMPUTE <n> = <expression>.

关键字 COMPUTE可选。

4.1 基本算术运算

ABAP/4 支持四种基本算术运算，同时还支持乘方计算。可以用数学表达式指定以下算

术运算符：

+	加法
-	减法
*	乘法
/	除法
DIV	整除
MOD	求余
**	求幂

可以用关键字 ADD、SUBTRACT、MULTIPLY 和 DIVIDE 进行基本算术运算，而不使用数学表达式中的运算符。

4.2 字符串的算术运算

类似于用 MOVE-CORRESPONDING 语句在字段串之间复制值，可以用以下关键字，执行字段串的算术运算：

- ADD-CORRESPONDING
- SUBTRACT-CORRESPONDING
- MULTIPLY-CORRESPONDING
- DIVIDE-CORRESPONDING

对于所有同名字段串组件，ABAP/4 进行相应算术运算。然而，仅当所有相关组件都是数值型数据类型时，该运算才有意义。

4.2.1 添加字段顺序

除基本算术运算中所述的基本加法之外，ADD 语句有几个变体，用于添加字段顺序。例如，添加字段顺序并将结果赋给另一个字段：

```
ADD <n1> THEN <n2> UNTIL <nz> GIVING <m>.
```

如果 <n1>、<n2>、 ... 、<nz>是在内存中相同类型和长度的等距字段序列，则进行求和计算并将结果赋给 <m>

添加字段顺序并将结果添加到另一个字段内容中：

```
ADD <n1> THEN <n2> UNTIL <nz> TO <m>.
```

该语句除了将字段总和添加到 <m>的旧内容中之外，与上面语句的工作方式相同。

4.3 数学函数

以下内部函数使用所有三种数值数据类型（ F、 I 和 P）作为参数。

函数	结果
ABS	参数的绝对值
SIGN	参数符号： 1 X > 0 SIGN(X) = 0 if X = 0

	-1 X < 0
CEIL	不小于参数的最小整数值
FLOOR	不大于参数的最大整数值
TRUNC	参数的整数部分
FRAC	参数的分数部分

4.4 浮点函数

以下内部函数用浮点数据类型（ F）作为参数。

函数	解释
ACOS ASIN 、 ATAN； COS SIN、 TAN	三角函数
COSH SINH 、 TANH	反三角函数
EXP	基数为 e 的求幂函数（ e=2.7182818285 ）
LOG	基数为 e 的自然对数
LOG10	基数为 10 的对数
SQRT	平方根

对于所有函数，正常数学约束适用。否则，会产生运行错误。

五字符串

5.1 移动字符串

5.1.1 按给定位置数移动字符串

要按给定位置数移动字段内容，使用 SHIFT 语句，用法如下：

SHIFT <c> [BY <n> PLACES] [<mode>].

该语句将字段 <c>移动 <n>个位置。 如果省略 BY <n> PLACES 则将 <n>解释为一个位置。如果 <n>是 0 或负值，则 <c>保持不变。如果 <n>超过 <c>长度，则 <c>用空格填充。 <n>可为变量。

对不同（ <mode>）选项，可以按以下方式移动字段 <c>：

LEFT: 向左移动 <n>位置，右边用 <n>个空格填充（默认设置） 。

RIGHT: 向右移动 <n>位置，左边用 <n>个空格填充。

CIRCULAR 向左移动 <n>位置，以便左边 <n>个字符出现在右边。

5.1.2 移动字符串到给定串位置

要移动字段内容以到给定串位置，则使用 SHIFT 语句，用法如下：

SHIFT <c> UP TO <str><mode>.

ABAP/4 查找 <c>字段内容直到找到字符串 <str> 并将字段 <c>移动到字段边缘。 <mode> 选项与按给定位置数移动字符串中所述相同。 <str> 可为变量。

如果 <c>中找不到 <str> ，则将 SY-SUBRC 设置为 4 并且不移动 <c>。否则，将 SY-SUBRC 设置为 0。

5.1.3 根据第一个或最后一个字符移动字符串

假设第一个或最后一个字符符合一定条件，则可用 SHIFT 语句将字段向左或向右移动。为此，请使用以下语法：

SHIFT <c>LEFT DELETING LEADING <str>.

SHIFT <c> RIGHT DELETING TRAILING <str>.

假设左边的第一个字符或右边的最后一个字符出现在 <str> 中，该语句将字段 <c>向左或向右移动。字段右边或左边用空格填充，<str> 可为变量。

5.2 替换字段内容

要用其它字符串替换字段串的某些部分，使用 REPLACE 语句。

REPLACE <str1> WITH <str2> INTO <c> [LENGTH <l>].

ABAP/4 搜索字段 <c>中模式 <str1> 前<l> 个位置第一次出现的地方。如果未指定长度，按全长度搜索模式 <str1> 。然后，语句将模式 <str1> 在字段 <c>中第一次出现的位置用字符串<str2> 替换。

如果指定长度 <l> ，则只替换模式的相关部分。如果将系统字段 SY-SUBRC 的返回代码设置为 0, 则说明在 <c>中找到 <str1> 且已用 <str2> 替换。非 0 的返回代码值意味着未替换。<str1> 、<str2> 和 <len> 可为变量。

5.3 大小写转换并替换字符串

可以将字母转换大 / 小写或使用替换规则。要转换大 / 小写，使用 TRANSLATE语句，用法如下：

TRANSLATE <c> TO UPPER CASE.

TRANSLATE <c> TO LOWER CASE.

这些语句将字段 <c>中的所有小写字母转换成大写或反之。

使用替换规则时，使用以下语法：

TRANSLATE <c> USING <r>.

该语句根据字段 <r>中存储的替换规则替换字段 <c>的所有字符。 <r>包含成对字母，其中每对的第一个字母用第二个字母替换。 <r> 可为变量。

5.4 转换为可排序格式

可以将字符字段转换为可按字母顺序排列的格式：

CONVERT TEXT <c> INTO SORTABLE CODE <sc>.

该语句为字符字段 <c>填充可排序目标字段 <sc>。字段 <c>必须是类型 C 且字段 <sc>必须是类型 X ，最小长度为 <c>长度的 16 倍。

该语句目的是为字符字段 <c>创建相关字段 <sc>, 作为 <c>的按字母顺序排列的排序关键字。在内表的内容和已解压缩的数据中进行排序。

如果对未转换的字符字段进行排序，则系统创建与各字母的特定平台内部编码相对应的顺序。在对目标字段进行排序之后，转换 CONVERTTEXT 按这样的方式创建目标字段，相应的字符字段顺序按字母排序。例如，在德语中，顺序为‘ Miller 、Moller 、Mller 、Muller ’，而不是‘ Miller 、 Moller 、 Muller 、 Mller ’。

转换方法依赖于运行 ABAP/4 程序的文本环境。文本环境在用户主记录中定义。例外的是可以使用如下语句，在程序中设置文本环境：

SET LOCALE LANGUAGE <lg> [COUNTRY <cy>] [MODIFIER <m>].

该语句根据语言 <lg> 设置文本环境。对于选项 COUNTRY, 只要特定国家语言不同，就可以在语言以外指定国家。对于选项 MODIFIER，只要一个国家内语言不同，就可以指定另一个标识符，例如，排序顺序在电话簿和词典之间不同。

字段 <lg>、<cy>和<m>必须是类型 C 且长度必须与表 TCP0C 的关键字段长度相等。表 TCP0C 是一个表格，从中进行平台相关的文本环境维护。在语句 SET LOCALE期间，系统根据 TCP0C 中的条目设置文本环境。除了内部传送的平台特性之外，用 SET 语句指定表关键字。如果 <lg> 等于 SPACE，则系统根据用户主记录设置文本环境。如果对于指定的关键字在表中无条目，则系统将产生运行错误。

文本环境影响 ABAP/4 中依赖于字符集的所有操作。

5.5 覆盖字符字段

要用另一字符字段覆盖字符字段，使用 OVERLAY语句，用法如下：

OVERLAY <c1> WITH <c2> [ONLY <str>].

该语句用 <c2>的内容覆盖字段 <c1>中包含 <str> 中字母的所有位置。<c2>保持不变。如果省略 ONLY <str> ，则覆盖字段 <c1>中所有包含空格的位置。

如果至少要替换 <c1>中的一个字符，则将 SY-SUBRC 设置为 0 。对于所有其它情况，将SY-SUBRC设置为 4 。如果 <c1>比<c2>长，则只覆盖 <c2>中的长度。

5.6 搜索字符串

要搜索特定模式的字符串，请使用 SEARCH语句，用法如下：

SEARCH <c> FOR <str><options>.

该语句在字段 <c>中搜索 <str> 中的字符串。如果成功，则将 SY-SUBRC 的返回代码值设

置为 0 并将 SY-FDPOS 设置为字段 <c>中该字符串的偏移量。否则将 SY-SUBRC 设置为 4。

搜索串 <str> 可为下列格式之一：

<str>	目的
<pattern>	搜索 <pattern> （任何字符顺序） 。忽略尾部空格
.<pattern>.	搜索 <pattern> ，但是不忽略尾部空格
*<pattern>	搜索以 <pattern> 结尾的词
<pattern>*	搜索以 <pattern> 开始的词

单词之间用空格、逗号、句号、分号、冒号、问号、叹号、括号、斜杠、加号和等号等分隔。

搜索字符字段 <c>的各种选项（ <options> ）如下

（1） ABBREVIATED

在字段 <c>中搜索包含 <str> 中指定字符串的单词, 其中字符可能被其它字符隔开。 单词和字符串的第一个字母必须相同。

（2） STARTING AT <n1>

在字段 <c>中搜索从 <n1>开始的 <str> 。结果 SY-FDPOS 参照相对于 <n1>的偏移量而不是字段的开始。

（3） ENDING AT <n2>

在字段 <c>搜索 <str> 直到位置 <n2>。

（4） AND MARK

如果找到搜索串，则将搜索串中的所有字符（和使用 ABBREVIATED时的所有字符）转换为大写形式。

5.7 获得字符串长度

要决定字符串到最后一个字符而不是 SPACE 的长度，使用内部函数 STRLEN, 用法如下：

[COMPUTE] <n> = STRLEN(<c>).

STRLEN将操作数 <c>作为字符数据类型处理, 而不考虑其实际类型。 不进行转换。 关键字 COMPUTE可选。

5.8 压缩字符串去空格

要删除字符字段中多余空格，使用 CONDENSE语句，用法如下：

CONDENSE <c> [NO-GAPS].

该语句去除字段 <c>中的前导空格并用一个空格替换其它空格序列。 结果是左对齐单词, 每个单词用空格隔开。如果指定附加的 NO-GAPS, 则去除所有空格。

5.9 连接字符串

要将单个字符串连接成一体，使用 CONCATENATE语句，用法如下：

CONCATENATE <c1> ... <cn> INTO <c> [SEPARATED BY <s>].

该语句连接字符串 <c1>与<cn>并将结果赋给 <c>。

该操作忽略尾部空格。附加 SEPARATED BY <s>允许指定字符字段 <s>，它放置在单个字段间已定义的长度中。

如果结果符合 <c>，则将 SY-SUBRC 设置为 0。然而，如果结果必须被截断，则将 SY-SUBRC 设置为 4。

5.10 拆分字符串

要将字符串拆分成两个或更多小串，使用 SPLIT 语句，用法如下：

SPLIT <c> AT <del> INTO <c1> ... <cn>.

该语句在字符字段 <c>搜索分界字符串 <del>，并将分界符之前和之后的部分放到目标字段<c1> ... <cn> 中。

要将所有部分放到不同目标字段中，必须指定足够目标字段。否则，用字段 <c>的剩余部分填充最后目标字段并包含分界符。

如果所有目标字段足够长且不必截断任何部分，则将 SY-SUBRC 设置为 0。否则，将其设置为 4。

5.11 分配字符串部分

MOVE语句的以下变体只使用类型 C 字段：

MOVE <c1> TO <c2> PERCENTAGE <p> [RIGHT].

将左对齐的字符字段 <c1>的百分比 <p>（或如果用 RIGHT 选项指定，则为右对齐）复制到<c2>。

<p>值可为 0 和 100 之间的任何数。将要从 <f1> 复制的长度取整为下一个整数。

如果语句中某一参数不是类型 C，则忽略参数 PERCENTAGE。

六逻辑流控制

6.1 比较运算符

要比较所有的字段类型，可以在逻辑表达式中使用下列运算符：

<运算符 >	含义
EQ	等于
=	等于
NE	不等于
<>	不等于
><	不等于
LT	小于

<	小于
LE	小于等于
<=	小于等于
GT	大于
>	大于
GE	大于等于
>=	大于等于

操作数可以是数据库字段、内部字段、文字或常数。

除基本字段外， 还可以将结构数据类型和上表中的运算符结合起来作为操作数。 字段串逐个组件进行比较，嵌套的结构分为基本的字段。

如果有意义， 可以对不同数据类型的字段进行比较。 如果字段可转换， 就可以进行比较。在比较之前，系统将按照下列层次规则执行自动类型转换：

（1）如果操作数之一是浮点数（类型 F），则系统将其它操作数转换为类型 F。

（2）如果操作数之一是压缩字段（类型 P），则系统将其它操作数转换为类型 P。

（3）如果操作数之一是日期字段（类型 D）或时间字段（类型 T），则系统将其它操作数转换为类型 D 或 T。不支持日期和时间字段之间的比较，这会导致程序中断。

（4）如果操作数之一是字符字段（类型 C）且其它操作数是十六进制字段（类型 X），则系统将类型 X 的操作数转换为类型 C。

（5）如果操作数之一是字符字段（类型 C），其它操作数为数字字段（类型 N），则系统将这两种操作数都转换为类型 P。

6.2 比较字符串和数字串

要比较字符串 （类型 C）和数字文本 （类型 N），可以在逻辑表达式中使用下列运算符。

<运算符 >	含义
CO	仅包含
CN	不仅包含
CA	包含任何
NA	不包含任何
CS	包含字符串
NS	不包含字符串
CP	包含模式
NP	不包含模式

因为除类型 N 和 C 外，系统不能执行任何其它类型转换， 所以， 在进行包含这些运算之一的比较时，操作数应该是类型 N 或 C。

运算符的功能如下：

（1） CO (仅包含)

如果 <f1> 仅包含 <f2> 中的字符，则逻辑表达式 <f1> CO <f2> 为真。该比较区分大小写，并包括尾部空格。如果比较结果为真，则系统字段 SY-FDPOS 包括 <f1> 的长度。如果为假，则 SY-FDPOS 包含 <f1> 中第一个未在 <f2> 内出现的字符的偏移量。

（2）CN (不仅包含)

如果 <f1> 还包含 <f2> 之外的其他字符，则逻辑表达式 <f1> CN<f2> 为真。该比较区分大小写，并包括尾部空格。如果比较结果为真，则系统字段 SY-FDPOS 包含 <f1> 中第一个未同时在 <f2> 中出现的字符的偏移量。如果为假，SY-FDPOS包含 <f1> 的长度。

（3）CA (包含任何)

如果 <f1> 至少包含 <f2> 的一个字符，则逻辑表达式 <f1> CA<f2> 为真。该比较区分大小写。如果比较结果为真，则系统字段 SY-FDPOS 包含 <f1> 中第一个也在 <f2> 中出现的字符的偏移量。如果为假，SY-FDPOS包含 <f1> 的长度。

（4）NA (不包含任何)

如果 <f1> 不包含 <f2> 的任何字符，则逻辑表达式 <f1> NA<f2> 为真。该比较区分大小写。如果比较结果为真，则系统字段 SY-FDPOS 包含 <f1> 的长度。如果为假，则 SY-FDPOS 包含 <f1> 中在 <f2> 内出现的第一个字符的偏移量。

（5）CS (包含字符串)

如果 <f1> 包含字符串 <f2>，则逻辑表达式 <f1> CS<f2> 为真。忽略尾部空格并且比较不区分大小写。如果比较结果为真，则系统字段 SY-FDPOS 包含 <f2> 在<f1> 中的偏移量。如果为假，SY-FDPOS包含 <f1> 的长度。

（6）NS (不包含字符串)

如果 <f1> 不包含字符串 <f2>，则逻辑表达式 <f1> NS<f2> 为真。忽略尾部空格且比较不区分大小写。如果比较为真，系统字段 SY-FDPOS 包含 <f1> 的长度。如果为假，系统字段 SY-FDPOS包含 <f2> 在 <f1> 中的偏移量。

（7）CP (包含模式)

如果 <f1> 包含模式 <f2>，则逻辑表达式 <f1> CP<f2> 为真。如果 <f2> 属于类型 C，则可以在 <f2> 中使用下列通配符：

15 * 用于任何字符串

15 + 用于任何单个字符

忽略尾部空格且比较不区分大小写。 如果比较结果为真， 系统字段 SY-FDPOS 包含 <f2> 在<f1> 中的偏移量。如果为假， SY-FDPOS包含 <f1> 的长度。

如果要对 <f2> 中的特殊字符进行比较， 请将换码符 # 放到其前面。可以使用换码字符 #

指定：

16　大小写字符

16　通配符 "*"(　输入 #*)

16　通配符 "+" (　输入 #+)

16　换码符号本身　(输入 ##)

16　字符串结尾的空格　(输入 #___)

（8） NP (不包含模式)

如果 <f1> 不包含模式 <f2> ，则逻辑表达式 <f1> NP<f2> 为真。 在 <f2> 中，可以使用与 CP 相同的通配符和换码字符。 忽略尾部空格且比较不区分大小写。 如果比较结果为真， 则系统字段 SY-FDPOS 包含 <f1> 的长度，如果为假， SY-FDPOS包含 <f2> 在 <f1> 中的偏移量。

6.3 比较二进制位结构

要将逻辑表达式初始操作数第一字节的二进制位结构与第二操作数的进行比较， 请使用下列操作符。

<运算符 >	含义
O	二进制位是 1
Z	二进制位是 0
M	混合二进制位

第二操作数的长度应为一个字节。

没有必要对第二操作数使用长度为 1 的十六进制字段 （类型 X），但却较方便， 因为其长度为一个字节且数字值直接与二进制位结构相关。

操作符功能如下：

O (二进制位是 1)

如果 <hex>中二进制位是 1 的位置，在 <f> 中是 1 ，则逻辑表达式 <f> O <hex> 为真。

Z (二进制位是 0)

如果 <hex>中二进制位是 1 的位置，在 <f> 中是 0 ，则逻辑表达式 <f> Z <hex> 为真。

M (混合二进制位)

如果从 <hex>中二进制位是 1 的位置起， <f> 中至少一个是 1 ，一个是 0 ，则逻辑表达式<f> M <hex> 为真。

6.4 编程分支和循环

6.4.1 IF 条件分支

IF 语句允许依据条件将程序流转到特定的语句块中。该语句块包括 IF 语句及其后面的 ELSEIF 、 ELSE 或 ENDIF 之间的所有命令。

IF <condition1>.

<statement block>

ELSEIF <condition2>.

<statement block>

ELSEIF <condition3>.

<statement block>

.....

ELSE.

<statement block>

ENDIF.

如果第一个条件是真，系统将执行所有语句直到第一个语句块结束，然后继续处理 ENDIF 语句之后的程序。 要采用选择性条件， 可以使用 ELSEIF 语句。如果第一个条件是假，系统使用与 IF 语句一样的方式处理后面的 ELSEIF 语句。如果 IF 或 ELSEIF 条件都为假，则执行 ELSE 开始的语句块。最后的语句块必须用 ENDIF 结束。

要用公式表达 IF 和 ELSEIF 语句中的条件， 可以使用任何编程逻辑表达式描述的逻辑表达式。

6.4.2CASE 条件分支

要根据特殊数据字段的内容执行不同的语句块，可以如下使用 CASE 语句：

CASE <f>.

WHEN <f1>.

<statement block>

WHEN <f2>.

<statement block>

WHEN <f3>.

<statement block>

WHEN ...

.....

WHEN OTHERS.

<statement block>

ENDCASE.

系统执行 WHEN 语句之后的语句块， 如果 <f> 内容等于 <fi> 的内容，且继续处理 ENDCASE

语句后面的语句。如果 <f> 的内容不等于 <fi> 的内容，则执行选项 WHEN OTHERS 后面的语句块。最后的语句块必须用 ENDCASE 结束。

6.4.3 DO 无条件循环

如果想要多次执行语句块，则可以如下使用 DO 语句编程循环：

```
DO [<n> TIMES] [VARYING <f> FROM <f1> NEXT <f2>].
<statement block>
ENDDO.
```

在发现 EXIT 、 STOP 或 REJECT 语句之前，系统继续执行由 DO 引导、 ENDDO 结束的语句块。

可以使用 TIMES 选项限制循环次数。 <n>可以是文字或变量。 如果 <n>是 0 或负数， 系统不执行该循环。

系统字段 SY-INDEX 中包含已处理过的循环次数。

6.4.4 WHILE 条件循环

如果只要条件为真，就不止一次执行语句，可以如下使用 WHILE 语句编程：

```
WHILE <condition> [VARY <f> FROM <f1> NEXT <f2>].
        <statement block>
ENDWHILE.
```

只要 <condition> 是真， 或系统发现 EXIT 、 STOP 或 REJECT 语句， 系统将继续执行由 WHILE 语句引导、 ENDWHILE 结束的语句块。

对于 <condition> ，可以使用编程逻辑表达式中描述的任何逻辑表达式。

系统字段 SY-INDEX 中包含已执行的循环次数。

可以任意嵌套 WHILE 循环，也可与其他循环结合使用。

WHILE 语句的 VARY 选项与 DO 循环的 VARYING 选项工作方式一样。允许每次执行循环时为变量 <f> 重新赋值。 <f1> 、<f2> 、<f3> 、 ... 必需是内存中类型相同和长度相等的一系列等距字段。第一次循环时，将 <f1> 分配给 <f> ，第二次循环时，将 <f2> 分配给 <f> ，以此类推。可以在一个 WHILE 语句中使用多个 VARY 选项。

6.5 终止循环

要终止循环过程，请使用下列关键字之一。

关键字	用途
CONTINUE	无条件终止循环过程
CHECK	有条件终止循环过程
EXIT	完全终止循环

在循环中只能使用 CONTINUE，但在循环外还可使用关键字 CHECK 和 EXIT，分别完成不同的功能。

6.5.1 无条件终止

要立即无条件终止循环，请如下使用 CONTINUE 语句：

```
CONTINUE.
```

在 CONTINUE语句之后，系统跳过当前语句块中所有剩余语句块，继续该语句后面的循环。

6.5.2 有条件终止

要有条件终止循环过程，请如下使用 CHECK 语句：

```
CHECK <condition>.
```

如果条件是假，系统跳过当前语句块中所有剩余语句块，继续后面的循环过程。对于 <condition> ，可使用编程逻辑表达式中描述的任何逻辑表达式。

6.5.3 完全终止

要无条件完全终止循环，请如下使用 EXIT 语句：

```
EXIT.
```

EXIT 语句之后，系统立即退出循环，继续结束语句 (ENDDO、ENDWHILE、ENDSELECT) 后面的处理。在嵌套循环中，系统仅退出当前循环。

七内表

7.1 创建内表

```
TYPES: BEGIN OF <f> OCCURS <n>,
<component declaration>,
        .....
END OF <f>.
DATA: <itab> TYPE STANDARD OF <f>,
      <wa> TYPE <f>.
```

其作用与先创建字段串 <f>，然后再创建与该字段串行结构相同的内表 <f> 相同。 <n>指定行的初始号。将第一行写入用类型 <t> 创建的内表数据对象之后，就为指定行保留内存。如果添加到内表中的行比 <n>指定的要多，则自动扩展保留的内存。如果内存中没有足够空间可用于内表，则将其写入缓冲区或磁盘（分页区域） 。

7.2 填充内表

要填充内表，既可逐行添加数据，也可复制另一个表格的内容。要逐行填充内表，可以使用 APPEND、COLLECT 或 INSERT 语句。

要将内表仅用于存储数据，出于性能方面的考虑，建议使用 APPEND。用 APPEND 也可以创建序列清单。

7.2.1 无关键字附加行

要将行附加到内表中，使用 APPEND 语句，用法如下：

```
APPEND [<wa> TO|INITIAL LINE TO] <itab>.
```

该语句将新行附加到内表 <itab> 中。

通过使用 <wa> TO 选项，指定要附加的源区域 <wa>。对于带表头行的表格，可以忽略 TO 选项。这样，表格工作区域就成了源区域。

可以使用 INITIAL LINE TO 选项替代 <wa> TO，将用其类型的正确值初始化的行添加到表格中。

APPEND 不考虑是否存在标准关键字相同的行。这样，可能会出现相同条目。系统字段 SY-TABIX 在每个 APPEND 语句之后包含附加行的索引。

7.2.2 根据标准关键字附加行

要用有唯一标准关键字的行填充内表，使用 COLLECT 语句，用法如下：

```
COLLECT [<wa> INTO] <itab>.
```

该语句通过使用 INTO 选项指定想附加的源区域 <wa>。如果表格有表头行，则可以忽略 INTO 选项。这样，表格工作区域就成了源区域。

系统检查表格条目的标准关键字是否相同。如果没有，COLLECT 语句的作用与 APPEND 语句相似，并将新行添至表格中。

如果存在关键字相同的条目，COLLECT 语句不附加新行，但将工作区域中数字字段的内容添加到现有条目中数字字段的内容中。系统字段 SY-TABIX 包含处理过的行的索引。

为 COLLECT 指定的工作区域必须与内表的行类型兼容，不仅仅是可转换为内表的行类型。COLLECT 语句无法用于带深层结构的内表，例如，将内表作为组件的行。

如果仅使用 COLLECT 语句填充内表，则不会出现重复条目。因此要填充没有重复条目的内表，应该使用 COLLECT 而不是 APPEND 或 INSERT。

7.2.3 插入行

要在内表行之前插入新行，使用 INSERT 语句，用法如下：

```
INSERT [<wa> INTO|INITIAL LINE INTO] <itab> [INDEX <idx>].
```

该语句通过使用 INTO 选项指定想插入的源区域 <wa>。如果表格有表头行，则可以忽略

INTO 选项。这样，表格工作区域就成了源区域。

可以使用 INITIAL LINE TO 选项替代 <wa> TO，将用其类型的正确值初始化的行添至表格中。

如果使用 INDEX 选项，则将新行插入到有索引 <idx> 的行之前。插入之后，新条目索引为<idx>，下行索引加 1。

如果表格包含 <idx> - 1 条目，系统将新条目附加到最后的现有表格行之后。如果表格的条目小于 <idx> - 1 ，系统无法插入条目并将 SY-SUBRC 设置为 4。如果操作成功，将SY-SUBRC设置为 0。

如果使用不带 INDEX 选项的 INSERT 语句，系统只能在 LOOP - ENDLOOP 循环内通过在当前行（例如带 SY-TABIX 返回索引的行）前插入新条目来处理它。

7.2.4 附加内表行

要将部分或全部内表附加到另一个内表中，使用 APPEND语句，用法如下：

```
APPEND LINES OF <itab1> [FROM <n1>] [TO <n2>] TO <itab2>.
```

如果没有 FROM 和 TO 选项，该语句将整个表格 ITAB1 附加到 ITAB2 中。如果使用这些选项，则可通过索引 <n1>或<n2>指定 ITAB1 中要附加的第一或最后一行。

用该方式将表格行附加到另一个表格中的速度比在循环中逐行进行附加快 3 到 4 倍。在APPEND语句之后，系统字段 SY-TABIX 包含附加的最后一行的索引。

7.2.5 插入内表行

要将部分或全部内表插入到另一个内表中，使用 INSERT 语句，用法如下：

```
INSERT LINES OF <itab1> [FROM <n1>] [TO <n2>]
        INTO <itab2> [INDEX <idx>].
```

如果没有 FROM 和 TO 选项，该语句将整个表格 ITAB1 附加到 ITAB2 中。如果使用这些选项，则可通过索引 <n1>或<n2>指定 ITAB1 中要附加的第一或最后一行。

如果使用 INDEX 选项，将 <itab1> 的行插入到 <itab2> 中索引为 <idx> 的行之前。如果不使用 INDEX 选项，系统只能在 LOOP - ENDLOOP 块中通过在当前行（例如，其索引在SY-TABIX 中返回的行）之前插入新条目来处理它。

7.2.6 复制内表

如果想一次将内表的全部内容复制到另一内表中，请使用 MOVE 语句或赋值操作符 (=) ，用法如下：

```
MOVE <itab1> TO <itab2>.
```

该语句等价于： <itab2> = <itab1>. 也可进行多重赋值，例如， <itab4> = <itab3> =

<itab2> = <itab1>. 也是可能的。 ABAP/4 从右到左进行处理： <itab2> = <itab1>.<itab3> = <itab2>.<itab4> = <itab3>. 这些语句执行完整操作。复制整个表格内容，包括作为表格组件的任何其它内表的数据。覆盖目标表格原来的内容。

对于有表头行的表格， 表格工作区域和表格本身同名。 要在上述语句中进行区分， 必须在名称之后输入两个方括号 ([]) 来定位内表而不是表格工作区域。

7.3 读取内表

7.3.1 逐行读取内表

要将内表逐行读入工作区域，可以使用 LOOP 语句编一个循环。语法如下所示：

```
LOOP AT <itab> [INTO <wa>] [FROM <n1>] [TO <n2>]
                [WHERE <condition>].
  .....
ENDLOOP.
```

用 INTO 选项指定目标区域 <wa>。如果表格有表头行，则可以忽略 INTO 选项。这样，表格工作区域就成了目标区域。

逐行将内表 <itab> 读入 <wa>或表格工作区域 <itab> 。对于读取的每一行， 系统都处理以LOOP开始，以 ENDLOOP结束的语句块。 可以用控制关键字 AT 在 LOOP - ENDLOOP 块内控制语句块流。

在语句块内， 系统字段 SY-TABIX 包含当前行的索引。 处理完表格的所有行之后循环结束。在 ENDLOOP语句之后，如果至少读取了一行，则将系统字段 SY-SUBRC 设置为 0。否则，将其设置为 4。

可以使用 FROM、 TO 或 WHERE选项限制要在循环中进行处理的行数。

22 使用 FROM 选项，可以用 <n1>指定要读取的第一行。

22 使用 TO 选项，可以用 <n2>指定要读取的最后一行。

22 用 WHERE选项， 可以指定 <condition> 的任何逻辑表达式。 第一个操作数必须是内表行结构的组件。 如果在循环中使用控制关键字 AT ，则不能使用 WHERE选项。

FROM和 TO 选项限制系统必须读取的行数。 WHERE选项仅避免对工作区域进行不必要的填充。用 WHERE选项，系统必须读取所有行。 为了提高效率， 应该尽可能使用 FROM 和 TO 选项。在某些条件下用 EXIT 语句而不是 WHERE选项跳出循环也十分有效。

7.3.2 用索引读取单行

要用索引从内表中读取单行，使用 READ 语句，用法如下：

```
READ TABLE <itab> [INTO <wa>] INDEX <idx>.
```

用 INTO 选项指定目标区域 <wa>。如果表格有表头行， 可以忽略 INTO 选项。 这样， 表格工作区域就成了目标区域。

系统用索引 <idx> 从表格 <itab> 中读取行。这比用关键字访问表格要快。

如果找到有指定索引的条目，则将系统字段 SY-SUBRC 设置为 0，而且 SY-TABIX 包含该行的索引。否则， SY-SUBRC包含非 0 值。

如果 <idx> 小于或等于 0 ，则 会发生实时错误。 如果 <idx> 超过表格大 小，系统将 SY-SUBRC中的返回代码值设置为 4 。

7.3.3 读取自定义关键字的单行

要从有自定义关键字的内表中读取单行， 使用 READ 语句的 WITH KEY 选项，用法如下：

READ TABLE <itab> [INTO <wa>] WITH KEY <key> [BINARY SEARCH].

用 INTO 选项可以指定目标区域。如果表格有表头行，则可以忽略 INTO 选项。这样，表格工作区域就成了目标区域。

系统读取 <itab> 中匹配 <key>中所定义的关键字的第一个条目。 如果找到有适当关键字的条目，则将系统字段 SY-SUBRC 设置为 0，并且 SY-TABIX 包含该行的索引。否则，将 SY-SUBRC设置为非 0 值。

如下所述，可以定义多个关键字 <key>：

（1）定义一系列关键字段

要定义自己的一系列关键字段，使用 WITH KEY 选项，用法如下：

....WITH KEY <k1> = <f1> ... <kn> = <fn> ...

自定义关键字包含表格组件 <k1>...<kn> 。字段 <f1>...<fn> 是关键字段的内容必须匹配的值。如果 <fi> 的数据类型与数据类型 <ki> 不兼容， 则<fi> 转换为类型 <ki> 。可以用 (<ni>) 代替 <ki> 来实时设置关键字段。 关键字段是字段 <ni> 的内容。 如果在运行时 <ni> 为空， 则系统忽略该关键字段。如果 <ni> 包含无效的组件名称，则发生实时错误。

用户可以为任何在关键字中使用的组件指定偏移量和长度。

（2）将整行定义为关键字

通过使用 WITH KEY 选项可将内表整行定义为其关键字，如下所示：

....WITH KEY = <value> ...

如果 <value> 的数据类型与表格行的数据类型不兼容， 则将 <value> 转换为表格行的数据类型。

对于此类关键字， 也可以选择由某个基本数据类型或内表直接定义的， 而不是由字段串直接定义的特定内表行。

（3）将行首定义为关键字

要将内表的行首定义为关键字，使用 WITH KEY 选项，用法如下：

....WITH KEY <k> ...

系统将（左对齐）的行首与 <k>进行比较。 <k>不能包含内表或包含内表的结构。 与上面两个选项不同之处在于用 <k>的数据类型进行比较。

7.3.4 读取标准关键字的单行

要从内表中读取有特定标准关键字的第一行，使用 READ 语句，用法如下：

READ TABLE <itab> [INTO <wa>] [BINARY SEARCH].

用户必须指定要从 <itab> 的表格工作区域中读取行的关键字。

读取语句的该变式只能用于有表头行的内表。

系统在表格中搜索第一个条目以匹配表格工作区域中的所有标准关键字段并将该行读入表格工作区域。如果使用 INTO 选项，则将该行读入工作区域 <wa>。

标准关键字包含内表关键字中所述的全部关键字段， 其中不包含 SPACE。如果找到有匹配关键字的条目， 则将系统字段 SY-SUBRC 设置为 0 并且 SY-TABIX 包含该行的索引。 否则，将 SY-SUBRC 设置为 4。

7.3.5 二分法搜索

用关键字读取单行时， 可以执行二分法搜索以代替标准顺序搜索。 为此， 使用 READ 语句的二分法搜索选项。

READ TABLE <itab> .....BINARY SEARCH.

如果使用二分法搜索选项，则必须按关键字中指定的次序对内表进行排序。

如果系统找到匹配指定关键字的多行，则读取索引最低的行。

二分法搜索比线性搜索要快。 因此，应尽可能将内表排序并且使用二分法搜索选项。

7.3.6 比较单行的内容

要将使用 READ 语句读取的单行内容与目标区域的内容进行比较， 可使用 READ 语句的 COMPARING 选项，用法如下：

READ TABLE <itab> [INTO <wa>] <key-option> COMPARING <fields>.

系统读取由关键字或 <key option> 中的索引指定的单行。读取行之后，将 <fields> 中指定的组件与目标区域中的相应组件进行比较。 可以用 INTO 选项指定目标区域 <wa>。如果表格有表头，则可以忽略 INTO 选项。这样，表格工作区域就成了目标区域。

对于 <field> ，可以编写一系列组件 ... <f1> ...<fn> 。也可以用 ... ALL FIELDS 指定所有组件。

如果系统找到 包含指定 <key-option> 的条目 ，且进行 比较的字 段内容相 同，则将SY-SUBRC设置为 0。如果进行比较的字段内容不同，则返回值 2 。如果系统找不到条目，则包含 4 。

如果系统找到条目，则无论比较结果如何，都将其读入目标区域。

7.3.7 读取一行部分列内容

要读取一行部分列的内容，使用 READ 语句的 TRANSPORTING选项，用法如下：

```
READ TABLE <itab> [INTO <wa>] <key-option> TRANSPORTING <fields>.
```

系统读取由关键字或 <key option> 中索引指定的单行。读取行之后，将 <fields> 中指定的组件传输给目标区域。 可以使用 INTO 选项指定目标区域 <wa>。如果表格有表头行， 可以忽略 INTO 选项。这样，表格工作区域就成了目标区域。

对于 <fields> ，可以用 ... <f1> ...<fn> 指定一系列组件。也可以用 ... NO FIELDS 指定不传输任何组件。

对于后一种情况， READ语句只影响系统字段 SY-SUBRC 和 SY-TABIX 。

7.3.8 确定内表属性

如果在处理过程中想知道内表一共包含多少行， 或者想知道定义的 OCCURS参数的大小，使用 DESCRIBE 语句，用法如下：

```
DESCRIBE TABLE <itab> [LINES <lin>] [OCCURS <occ>].
```

如果使用 LINES 参数， 则将填充行的数量写入变量 <lin> 。如果使用 OCCURS参数， 则将行的初始号写入变量 <occ>。

7.4 修改和删除内表行

7.4.1 用 MODIFY更改行

要用 MODIFY 语句更改行，使用：

```
MODIFY <itab> [FROM <wa>] [INDEX <idx>].
```

FROM选项中指定的工作区域 <wa>代替 <itab> 中的行。如果表格有表头行，可以忽略FROM选项。这样，表格工作区域就代替行。

如果使用 INDEX 选项， 则新行代替索 引为 <idx> 的现有 行。如果替换 成功，则将SY-SUBRC设置为 0。如果内表包含的行少于 <idx> , 则不更改任何行并且 SY-SUBRC 包含 4。

如果使用没有 INDEX 选项的 MODIFY 语句，则系统只能在 LOOP - ENDLOOP 块中通过更改当前行（例如由 SY-TABIX 返回其索引的行）来处理它。

7.4.2 用 WRITE TO更改行

要用 WRITE TO 语句更改行，请使用下列语法：

WRITE <f>[+<o1>][(<l1>)] TO <itab>[+<o2>][(<l2>)] INDEX <idx>.

将字段 <f> 中偏移量为 <o1>，长度为 <l1> 部分的内容复制到索引为 <idx> 的表格行中，覆盖偏移量为 <o2>，长度为 <l2> 的部分。请注意，即使对于有表头行的表格，带 INDEX 选项的 WRITE TO 语句也不访问表格工作区域，而是访问表格的某一行。

WRITE TO 语句不能识别表格行的结构。SAP 建议只在转换已知其确切位置的标志时才使用该语句。另一种情况是用一个基本字符字段定义的内表。该结构的表格非常重要，例如，用于程序的动态生成。

7.4.3 在循环中删除行

要在循环中从内表中删除行，使用 DELETE 语句，用法如下：

DELETE <itab>.

系统只能在 LOOP - ENDLOOP 块中处理该语句。删除第一行后，可以取消当前行的定义并取消其对 SY-TABIX 内容的赋值。要在该循环内进一步处理行，需使用有 INDEX 选项的语句。

7.4.4 用索引删除行

要使用索引删除行，使用有 INDEX 选项的 DELETE 语句，用法如下：

DELETE <itab> INDEX <idx>.

如果使用 INDEX 选项，则从 ITAB 中删除索引为 <idx> 的行。删除行之后，下面行的索引减 1。如果操作成功，则将 SY-SUBRC 设置为 0。否则，如果不存在索引为 <idx> 的行，则 SY-SUBRC包含 4 。

如果在 LOOP- ENDLOOP 块中删除某一条目，则当前行及其对 SY-TABIX 内容的赋值可成为未定义。要在该循环内进一步处理行，需使用有 INDEX 选项的语句。

7.4.5 删除邻近的重复条目

要删除邻近重复条目，使用 DELETE 语句，用法如下：

DELETE ADJACENT DUPLICATE ENTRIES FROM <itab> [COMPARING <comp>].

系统从内表 <itab> 中删除所有邻近重复条目。

如果没有 COMPARING 选项，则标准关键字段的内容必须相同。

如果有 COMPARING 选项 .... COMPARING <f1><f2>... , 指定字段 <f1><f2> ... 的内容必须相同。 也可以通过写入 (<name>) 代替 <f1> 在运行时在括号中指定字段名。字段 <name> 包含排序关键字段的名称。如果 <name> 在运行时为空，则系统将其忽略。 如果包含无效的组件名，则会发生实时错误。

如果有 COMPARING 选项 .... COMPARING ALL FIELDS，所有字段的内容必须相同。如果

系统找到并删除至少一个重复条目，则将 SY-SUBRC 设置为 0。否则，将其设置为 4。

如果表格根据指定的比较标准进行过排序， 则可使用该语句从内表中删除所有重复条目。

7.4.6 删除选定行

要删除一组选定行，使用 DELETE 语句，用法如下：

DELETE <itab> [FROM <n1>] [TO <n2>] [WHERE <condition>].

用户必须至少指定三个选项之一。 如果使用没有 WHERE选项的该语句， 则系统从 <itab> 中删除所有索引在 <n1>和 <n2>之间的行。如果不使用 FROM 选项，则系统从第一行开始删除。如果不使用 TO 选项，则系统删除所有行直到最后一行。

如果使用 WHERE选项，则系统仅从 <itab> 中删除满足条件 <condition> 的行。对于 <condition> ，可指定任何逻辑表达式。第一个操作数必须是内表行结构的组件。

如果系统至少删除一行，则将 SY-SUBRC 设置为 0。否则，将其设置为 4。

7.5 内表排序

要将内表排序，使用 SORT 语句，用法如下：

SORT <itab> [<order>] [AS TEXT]

[BY <f1> [<order>] [AS TEXT] ... <fn> [<order>] [AS TEXT]].

如果不使用 BY 选项，则根据其标准关键字对内表 <itab> 进行排序。 要定义不同的排序关键字，使用 BY 选项。系统就根据指定组件 <f1> ... <fn> 对数据集进行排序。这些字段可以是任何类型， 包括类型 P、I 和 F 字段，或者表格。 排序关键字段的数目限制在 250 以内。如果指定多个关键字段， 则系统首先根据 <f1>，然后根据 <f2>，以此类推对记录进行排序。系统使用 BY 之前指定的选项作为 BY 之后指定的所有字段的缺省选项。 在单个字段之后指定的选项覆盖选项在 BY 之前指定的这些字段。

如果在运行时排序标准仍然未知， 可以通过写入 (<name>) 代替 <fi> 进行动态设置 。字段 <name>包含排序关键字段的名称。 如果 <name>在运行时为空， 系统就将其忽略。 如果包含无效的组件名， 则发生实时错误。 对于任何在排序字段中使用的字段， 用户都可指定偏移量和长度。

用户可以通过在 <order> 选项中输入 DESCENDING或 ASCENDING来指定排序顺序。 标准顺序是升序。

用户可以用选项 AS TEXT 影响字符字段的排序方式。如果没有 AS TEXT，则系统二分排序字符字段并根据它们的平台相关内部编码。如果有选项 AS TEXT，系统根据当前文本环境按字母顺序排序字符字段。 用户可用语句 SET LOCALLANGUAGE设置文本环境， 这是例外。使用选项 AS TEXT，用户可免除在排序之前将字符字段转换为可排序格式之劳。此类转换仅

在下列情况下才有必要：

首先按字母顺序对内表进行排序，然后二分法进行搜索。按字母顺序排序后的内表次序与按二分法排序后的次序不同。

用字母关键字多次对内表进行排序。在这种情况下效率更佳，因为只进行一次转换。在程序中为数据库表格创建字母索引。

如果在 BY 之前指定 AS TEXT，则选项仅影响排序关键字中的字符字段。如果在字段名之后指定 AS TEXT，则该字段必须为类型 C。

如果自己指定排序关键字，通过使关键字相对短些可提高效率。但是，如果排序关键字包含内表，则排序进程可能会慢很多。

排序并不稳定。这意味着也许没有必要保留排序关键字相同的行的旧次序。

如果主内存中没有足够的空间用于排序，系统就将数据写入临时外部文件。该文件名在 SAP 参数文件参数 DIR_SORTTMP中定义。

7.6 创建顺序表

内表适合于生成顺序表。为此，从空的内表开始，然后使用 APPEND 语句的 SORTEDBY 选项，用法如下：

APPEND [<wa> TO] <itab> SORTED BY <f>.

如果使用有 SORTED BY 选项的 APPEND 语句，则并不将新行附加为内表 <itab> 的最后一行。而是系统插入新行，这样内表 <itab> 就根据字段 <f> 以降序排序。

要生成包含多达 100 个条目的顺序表，则应该使用 APPEND语句。在处理更大的表时，由于效率方面的原因，建议用 SORT 语句对表格进行排序。

如果使用 SORTED BY 选项，表格只能包含 OCCURS参数中指定的行数。这是一般规则的一个例外。如果添加的行数比指定的要多，则丢弃最后一行。这对于创建长度有限的次序表十分有用。

使用 APPEND 语句的 SORTED BY 选项指定的工作区域必须与内表的行类型兼容。可转换性对该选项不充分。

7.7 AT …… ENDAT行组控制级别

用控制级别语句 AT 可以打开语句块，用控制级别语句 ENDAT 可以关闭它。语法如下所示：

AT <line>.

<statement block>

ENDAT.

在处理 AT- ENDAT 内语句块的行条件 <line> 可以是：

<line>	含义
FIRST	内表的第一行
LAST	内表的最后一行
NEW <f>	行组的开头，与字段 <f> 和 <f> 剩余字段中的内容相同
END Of <f>	行组的结尾，与字段 <f> 和 <f> 剩余字段中的内容相同

AT - ENDAT块中的语句块使用这些行条件代表预定义的控制结构。用户可以使用它们处理内表中的控制断点，而不必使用编程分支和循环中所述的语句自己编程。

在 AT - ENDAT语句块中，工作区域没有用当前表格行进行填充。初始化所有不是标准关键字部件的字段。 对于行条件 FIRST 和 LAST, 系统用星号 (*) 改写所有标准关键字段。对于行条件 NEW<f> 和 END OF <f> ，系统用星号 (*) 改写所有出现在工作区域中指定字段 <f> 右边的标准关键字段。用户可根据自己的需求在 AT - ENDAT 语句块中填充工作区域。

使用 AT NEW f 和 AT END OF f 时需注意：

（1）f 必须是内表的第一个字段；

（2）内表中 f 之后的字段的值都会变成“ * ”。

使用 AT NEW, AT FIRST, AT LAST,AT END OF 时需注意： LOOP的时候不能加条件； AT 和 ENDAT之间不能使用 LOOPINTO 的 working aera 。手动实现 AT NEW,ATENDOF的时候，需要注意，尤其是在 AT END OF 的时候。

7.8 比较内表

可以将内表用做逻辑表达式的操作数：

.... <itab1><operator><itab2> ...

对于 <operator> , 可以使用比较所有的字段类型中的表格内列出的所有操作符 （EQ = 、NE <>、 ><、 GE、 >=、 LE 、 <=、 GT、 >、 LT 、 <）。

进行内表比较的第一个条件是它们包含的行数。内表包含的行数越多，则内表就越大。如果两个内表行数相同， 则逐行、 逐个组件进行比较。 如果表格行的组件本身就是内表， 则进行递归比较。 如果使用等于操作符以外的操作符， 则系统找到一对不相等的组件后就立即停止比较并返回该结果。

对于有表头行的内表， 则可在表格名之后使用方括号 [] 以将表格工作区域和表格体区别开来。

7.9 初始化内表

要初始化有或没有表头的内表，使用 REFRESH语句，用法如下：

REFRESH <itab>.

该语句将内表重置为填充它以前的状态。 这意味着表格将不包含任何行。 如果使用没有

表格工作区域的内表，可以使用 CLEAR 语句代替 REFRESH语句，用法如下：

CLEAR <itab>.

如果使用有表头行的内表， CLEAR语句仅清除表格工作区域。 要重置整个内表而不清除表格工作区域，使用 REFRESH语句或 CLEAR 语句，用法如下：

CLEAR <itab>[].

内表名称之后的方括号指内表体。

使用 REFRESH或 CLEAR 初始化内表后，系统保持在内存中保留的空间。可以用 FREE 语句释放内存，用法如下：

FREE <itab>.

也可以使用 FREE 语句重置内表并直接释放其内存， 而不必先使用 REFRESH或 CLEAR。与 REFRESH一样， FREE在表格体上，而不在表格工作区域上工作。

在 FREE 语句之后， 可以再次定位内表。 这样， 系统就再次保留内存空间。 可以使用如下逻辑表达式检查内表是否为空：

... <itab> IS INITIAL ...

八模块化

8.1 宏的定义和调用

要定义包含部分源代码的宏，使用 DEFINE 语句，用法如下：

DEFINE <macro>.

<statements>

END-OF-DEFINITION.

这就定义了宏 <macro>。必须在 DEFINE 和 END-OF-DEFINITION 之间指定完整的语句。这些语句最多可以包含九个占位符（ &1, &2, , , &9)。

完成宏定义之后，就可以进行调用，方法如下：

<macro> [<p1><p2> ... <p9>].

在生成程序期间，系统用已定义的语句替换 <macro>，用 <pi> 替换每个占位符 &i 。可以从宏中调用另一个宏，但宏不能调用自己。

8.2 使用包含程序

要从另一个 ABAP/4 程序中调用包含程序，使用 INCLUDE 语句，用法如下：

INCLUDE <incl>.

在语法检查期间和生成期间，该语句将源代码 <incl> 插入到 ABAP/4 程序中。 INCLUDE

语句的功能类似于把 <incl> 源代码复制到调用程序中语句的位置上。

INCLUDE语句必须单独占一行，并且不能扩展到多行。

包含程序不是运行时装载的，而是在程序生成前就被解析。程序生成后，则包含所用全部包含程序的源代码。

8.3 调用子程序

8.3.1 调用内部子程序

要调用内部子程序，使用 PERFORM语句，用法如下：

PERFORM <subr> [<pass>].

调用子程序 <subr> 。在 <pass> 选项中，指定如何与子程序进行数据传递。如果不使用 <pass> 选项，子程序也可以访问主 ABAP/4 程序中声明的所有数据类型和对象。该数据叫做全局数据，如果不被同名局域数据隐藏，该数据对子程序可见。

8.3.2 调用外部子程序

要调用外部子程序，使用 PERFORM语句，用法如下：

PERFORM <subr>(<prog>) [<pass>] [IF FOUND].

调用程序 <prog> 中定义的子程序 <subr> 。如果希望与子程序进行数据传递，必须定义 <pass> 选项或者使用公用部分。如果使用 IF FOUND 选项，并且程序 <prog> 中没有子程序 <sub>，系统就忽略 PERFORM语句。

启动调用外部子程序的程序时，如果定义了子程序的程序不在内存中，ABAP/4 就将其装载到内存中。为了节省存储空间，尽量将不同程序中定义的子程序数目限制到最小。

8.3.3 运行时指定子程序名

运行时，可以指定要调用的子程序名以及存储子程序的程序名。为此使用 PERFORM语句，用法如下：

PERFORM (<fsubr>) [IN PROGRAM (<fprog>)] [<pass>] [IF FOUND].

系统执行字段 <fsubr> 中存储的子程序。如果使用 IN PROGRAM 选项，系统在字段 <fprog> 中存储的程序内查找子程序 （外部调用）。否则，系统在当前程序中查找子程序 （内部调用）。

使用该语句还可在程序代码中指定子程序名和外部程序名。为此请忽略括弧。<pass> 选项指定如何与子程序进行数据传递。如果使用 IF FOUND 选项，找不到子程序 <sub>时，系统就忽略 PERFORM语句。

8.3.4 从列表中调用子程序

要从列表中调用特定子程序，使用 PERFORM语句，用法如下：

PERFORM <idx> OF <form1><form2> ....<formn>.

系统执行子程序列表中位于 <idx> 处的子程序。 PERFORM语句的这一变量只对内部调用有效。字段 <idx> 既可以是变量也可以是文字。

8.4 通过参数进行数据传递

可以用参数在调用程序和子程序之间进行数据传递。

在定义子程序期间用 FORM 语句定义的参数叫做形式参数。

在调用子程序期间用 PERFORM语句指定的参数叫做实参数。

可以区分不同种类的参数： 输入参数用于向子程序传递参数； 输出参数用于从子程序传递参数；输入 / 输出参数用于与子程序进行参数传递

在 FORM和 PERFORM语句的 <pass>选项中定义参数，方法如下：

```
FORM <subr> [TABLES <formal  table list>]
    [USING <formal  input list>]
  [CHANGING <formal output list>]....
PERFORM <subr>[(<prog>)] [TABLES <actual  table list>]
                [USING <actual  input list>]
              [CHANGING <actual output list>]....
```

选项 TABLES、 USING 和 CHANGING必须按上述顺序编写。

列表中 USING 和 CHANGING后面的参数可以是所有类型的数据对象和字段符号。列表中 TABLES 后面的参数既可以是有表头行的内表，也可以是不带表头行的内表。可以采用 TABLES、 USING 或 CHANGING传送内表。

对于 FORM语句中 USING 和 CHANGING后面列表中的每个形式参数，可以指定不同数据传递方式：

通过参考值调用： 在子程序调用期间， 仅将实参数的地址传送给形式参数。 形式参数本身没有内存。 在子程序中使用调用程序的字段。 更改形式参数之后， 调用程序中的字段内容也会变动。

通过值调用： 在子程序调用期间， 形式参数是作为实参数的副本创建的。 形式参数有自己的内存。更改形式参数并不影响实参数。

通过值和结果调用： 在子程序调用期间， 形式参数是作为实参数的副本创建的。 形式参数有自己的内存空间。在子程序末尾，将对形式参数的更改复制给实参数。

8.4.1 通过参考传递

要通过参考值在调用程序和子程序之间进行数据传递，使用 FORM 和 PERFORM语句 <pass>选项的 USING 或 CHANGING，用法如下：

FORM ..... [USING <fi1> ... <fin>] [CHANGING <fo1> ... <fon>] ...

PERFORM... [USING <ai1> ... <ain>] [CHANGING <ao1> ... <aon>] ...

在 USING 和 CHANGING后面的列表中指定形式参数和实参数，而不需附加任何内容。

FORM语句中形式参数的 名称可以与 PERFORM语句中实参数 <ai1> ... <ain> and <ao1> ... <aon> 的名称不同。 PERFORM语句列表中的第一个参数传递给 FORM 语句相应列表中的第一个参数，以此类推。

对于通过参考值调用， USING 和 CHANGING完全相同。 对于文档， USING 用于子程序中的固定输入参数，而 CHANGING则用于子程序中变化的输出参数。

8.4.2 通过值传递

要确保输入参数在调用程序中保持不变（即使子程序中已被更改） ，可以通过值将数据传递给子程序。为此，使用 FORM 和 PERFORM语句 <pass> 选项的 USING，用法如下：

FORM ..... USING ...VALUE(<fii>) ..

PERFORM... USING .....<aii> ..

通过对 FORM 语句中 USING 后面列表中的形式输入参数写入 VALUE(<fii>) 而不是 <fii> ，相应参数就通过值进行传递。 与实字段 <ai> 属性相同的 PERFORM语句调用该子程序时，就会创建一个新的局域字段 <fi> 。系统处理独立于调用程序中参考字段的该字段。

8.4.3 通过值和结果进行传递

如果仅希望在子程序运行成功之后，才将更改过的输出参数从子程序返回给调用程序，使用 FORM 和 PERFORM语句 <pass>选项的 CHANGING，用法如下：

FORM ..... CHANGING ...VALUE(<fii>) ..

PERFORM... CHANGING .....<aii> ..

通过对 FORM 语句中 CHANGING后面列表中的形式输入参数写入 VALUE(<fii>) 而不是 <fii> ，相应参数通过值和结果调用进行传递。 与实字段 <ai> 属性相同的 PERFORM语句调用该子程序时，就会创建一个新的局域字段 <fi> 。系统处理独立于调用程序中参考字段的字段。

仅在运行到 ENDFORM语句时，系统才将 <fii> 的当前值传递给 <aii> 。如果子程序因为某个对话信息而中断，则实参数 <aii> 保持不变。

8.5 在子程序中定义局部数据类型和对象

8.5.1 定义动态局部数据类型和对象

可以用 TYPES 和 DATA 语句，按照创建数据对象和数据类型中的说明，在子程序内创建局部数据类型和数据对象。 为每个子程序调用新近定义的这些类型和对象将在退出子程序时删除。

每个子程序均有自己的局域命名空间。　　如果用与全局数据类型或对象相同的名称定义局部数据类型或对象，　则不能在子程序中访问全局数据类型或对象。　　局部数据类型或数据对象会隐藏同名全局数据类型或对象。这意味着，如果在子程序中使用某个数据类型或对象名，则总是定址局部声明的对象（如果存在）　　，否则，定址全局声明的对象。

8.5.2　定义静态局部数据类型和对象

如果想在退出子程序后保留局部数据对象之值，必须使用　　STATICS 语句而非 DATA 语句进行声明。使用　STATICS 声明已全局定义，但仅在定义它的子程序中局部可见的数据对象。

8.5.3　显示定义全局数据对象

要避免全局数据对象值在子程序内被更改，使用　　LOCAL 语句，用法如下：

LOCAL <f>.

只能在 FORM 语句和 ENDFORM语句之间使用该语句。　对于 LOCAL ，可以保存无法被子程序内的数据声明隐藏的全局数据对象值。

8.6　调用功能模块

要从 ABAP/4 程序调用功能模块，使用　CALL 语句，用法如下：

CALL FUNCTION <module>

[EXPORTING f1 = a1 .... fn = an]

[IMPORTING f1 = a1 .... fn = an]

[CHANGING f1 = a1 .... fn = an]

[TABLES f1 = a1 .... fn = an]

[EXCEPTIONS e1 = r1 .... en = rn [OTHERS = ro]].

可以将功能模块<module>的名称指定为文字或变量。通过将实参数显式指定给EXPORTING、IMPORTING、TABLES 或 CHANGING选项后面列表中的形式参数，与功能模块之间进行参数传递。

分配总是有以下格式：　<formal parameter> = <actual parameter>

（1）EXPORTING选项允许将实参数　ai　传递给形式输入参数　fi　。在功能模块中，　必须将形式参数声明为输入参数。

（2）IMPORTING选项允许将形式输出参数　　fi　传递给实参数　ai　。在功能模块中，　必须将形式参数声明为输出参数。

（3）CHANGING选项允许将实参数　ai　传递给形式参数　fi　，并在处理功能模块后，系统将（更改过的）形式参数　　fi　传递回实参数　ai　。在功能模块中，必须将形式参数声明为

更改参数。

（4）TABLES 选项允许在实参数和形式参数间传递表格。 借助该选项， 内表总是通过参考值传递。

EXPORTING IMPORTING 和 CHANGING选项的参数可以是任意类型的数据对象。 这些选项的功能类似于子程序的 FORM 和 PERFORM语句中的 USING 和 CHANGING选项。 TABLES 选项的参数必须为内表。 TABLES选项对应于 FORM 和 PERFORM语句的 TABLES 选项。

用 EXCEPTIONS选项， 可以处理功能模块处理过程中发生的例外。 例外是功能模块的特殊参数。 有关如何定义和出现例外的内容， 创建和编程功能模块中有专门说明。 如果出现 ei 例外， 则系统停止执行功能模块， 并且不将任何值从功能模块传递给程序， 通过参考值传递的值例外。 如果在 EXCEPTION 选项中指定了 ei ，则调用程序通过将 ri 分配给 SY-SUBRC 来处理例外。必须将 ri 指定为数字文字。可以使用 EXCEPTION 列表中的 OTHERS处理列表中没有显式指定的所有例外，并且可将同一数值 ri 用于多个例外。

九数据库

9.1 开放式 SQL

关键字	用途
SELECT	从数据库表读取数据
INSERT	将行添加到数据库表
UPDATE	更改数据库表中的行
MODIFY	添加或删除行
DELETE	从数据库表删除行
OPEN CURSOR, FETCH, CLOSE CURSOR	用光标从数据库表中读取行
COMMIT WORK, ROLLBACK WORK	确认或撤消对数据库表的更改

在 ABAP/4 程序中使用开放式 SQL 语句时，必须保证：

（2）被定址的数据库系统必须是 SAP 支持的；

（2）必须已经在 ABAP/4 词典中定义了被定址的数据库表

下列系统字段在开放式 SQL 操作中起着重要的作用：

（1） SY-SUBRC

和使用其它 ABAP/4 语句一样， 系统字段 SY-SUBRC 中的返回代码值表示在每个开放式 SQL 操作之后该操作是否成功。如果操作成功， SY-SUBRC的值就等于 0 ；如果操作失败，SY-SUBRC的值就不等于 0 。

（2）SY-DBCNT

SY-DBCNT字段中的值表明受该操作影响的行数，或已被处理的行数。

9.2 读取数据

要从数据库表读取数据，使用 SELECT 语句：

SELECT <result> FROM <source> [INTO <target>] [WHERE <condition>]

[GROUP BY <fields>] [ORDER BY <sort_order>].

该语句有几个基本子句。下表中列出了每一个子句：

子句	说明
SELECT <result>	定义选择的结果
FROM <source>	指定将读取的数据库表
INTO <target>	选定数据指定目标区
WHERE <condition>	选择即将读取的行
GROUP BY <fields>	行分组
ORDER BY <sort_order>	指定行的顺序

9.2.1 定义选择结果 SELECT

SELECT子句定义是选择单行还是选择多行、是否去掉重复行以及将选择哪些列。

（1）选择多行中的所有数据

要从数据库表中读取所有列和多行，按如下方式在循环中使用 SELECT 语句。

SELECT [DISTINCT] * .....ENDSELECT.

必须用 ENDSELECT语句结束该循环。该循环依次读取所选行，并为每个读取的行执行循环中的 ABAP/4 语句。 SELECT循环的结果是与被读取的数据库表的格式完全相同的表。

DISTINCT 选项将自动的去掉重复的行。如果至少读取了一行，系统字段 SY-SUBRC 就返回 0 。如果没有读取， 系统字段 SY-SUBRC 就返回 4 。系统字段 SY-DBCNT 给读取的行计数。每执行一次 SELECT 语句， SY-DBCNT都加 1 。

（2）选择单行中的所有数据

要从数据库表中读取单个行的所有列，按如下方式使用 SELECT 语句：

SELECT SINGLE [FOR UPDATE] * ..... WHERE <condition> .....

该语句的结果是一个单行。为了保证清楚地指定了一行，就必须在 WHERE子句的条件 <condition> 中用 AND 链接形成数据库表主码的所有字段。

如果系统没有找到具有指定关键字的行， 系统字段 SY-SUBRC 将设置为 4 。如果系统找到了一个完全符合指定条件的行， SY-SUBRC就返回 0 。

可以使用 FOR UPDATE选项锁定在数据库表中选定的行。 程序将一直等到接收到该锁定。如果数据库检测到或怀疑有一个死锁定，将产生运行时间错误。

（3）选择指定列中的所有数据

要读取显式地包含规定的列的行，或要得到关于数据库表特定列的摘要信息，按如下方法使用 SELECT 语句：

SELECT [SINGLE [FOR UPDATE]] [DISTINCT] <s1><s2> .....

9.2.2 指定数据表 FROM

（1）在程序中指定数据库表

要在程序中指定即将读取的数据库表或视图，按下列格式使用 FROM 子句：

..... FROM <dbtab> [CLIENT SPECIFIED] [BYPASSING BUFFER]

[UP TO <n> ROWS].....

数据库表或视图 <dbtab> 必须对 ABAP/4 词典有效，并且必须在 ABAP/4 程序中包含相应的 TRABLES 语句。

CLIENT SPECIFIED 选项用于关闭自动集团处理，BYPASSING BUFFER 选项使得不用读取 SAP 表缓冲区就可直接读取数据库。

（2）在运行时指定数据库表

可以在运行时指定数据库表的名称。为此，按如下格式使用 FROM 子句：

.....FROM (<dbtabname>) [CLIENT SPECIFIED] [BYPASSING BUFFER]

[UP TO <n> ROWS].. INTO <target> .....

该格式的 FROM 子句只能与 INTO 子句一起使用。

字段 <dbtabname>的内容确定了数据库表的名称。在该情况下，程序不必包含 TABLES 语句。选项 CLIENT SPECIFIED、BYPASSING BUFFER 和 UP TO <n> ROWS 同样是用于在程序中指定数据库表的名称。

9.2.3 指定目标区 INTO

（1）将数据读到工作区

可以将数据从数据库表读到与 TABLES 语句中定义的默认工作区不同的工作区（通常是字段串）中。为此，按照如下格式在 SELECT 语句的 INTO 子句中指定工作区：

SELECT ... INTO <wa> .....

必须为工作区 <wa>声明一个至少与将读取的行一样大的数据对象。

（2）将数据读到内表

可以在单个操作中将数据库表中行选择的结果集写入内表中。为此，按照如下格式在SELECT语句的 INTO 子句中指定内表：

SELECT .....INTO TABLE <itab>.

在该情况下， SELECT并不启动循环， 并且不允许使用 ENDSELECT语句。如果内表 <itab> 不是空的，那么， SELECT语句将用读取的数据覆盖其中的内容。

（3）将数据读到预定义包大小的内表

如果需要将所选行按已预定义大小的包的形式读到内表中， 按如下格式使用 INTO 子句的 PACKAGE SIZE 选项：

SELECT * .....INTO TABLE <itab> PACKAGE SIZE <n>.....

该语句打开一个循环， 必须使用 ENDSELECT语句结束该循环。 对于读取的每个包含 <n> 行的包，系统都将执行一次循环。如果 <n>小于或等于零，将出现运行时间错误。

（4）将数据行附加到内表中

为了避免覆盖内表中的内容，可将所选行附加到此表中。为此，按如下格式使用 APPENDING子句，而不是 INTO 子句：

SELECT ..... APPENDING TABLE <itab>.....

与上面描述的 INTO 子句的唯一区别是它是将行附加到内表 <itab> 中，而不是替换。 也可以在该语句中使用 PACKAGE SIZE 选项。

（5）逐个组件的读取数据

要一个组件接一个组件地将数据读到目标区中，请使用 INTO 子句的 CORRESPONDING FIELDS 选项。语法如下所示：

对于将数据读到工作区中：

SELECT ... INTO CORRESPONDING FIELDS OF <wa> .....

对于将数据读到内表中：

SELECT ... INTO CORRESPONDING FIELDS OF TABLE <itab> .....

对于将数据附加到内表中：

SELECT ... APPENDING CORRESPONDING FIELDS OF TABLE <itab> .....

这些语句不会将所选行的所有字段放进目标区中。 系统只将列的内容传送到目标区的对应组件中。如果可能，在传送过程中，可将值转换成目标字段的数据类型。

9.2.4 选择读取行 WHERE

（1）在程序中为行选择指定条件

要在程序中为行选择指定条件，按如下格式使用 WHERE子句：

..... WHERE <condition> .....

有六个基本条件可用于限制行选择。描述如下：

1）<f><operator><g>

<f> 是不带作为前缀的表名称的数据库字段名　　（数据库表的列），<g>是任意字段或字母。字段名称和操作符必须用空格隔开。

2）<f> NOTBETWEEN <g1> AND <g2>

数据库字段 <f> 的值必须（不能）处于字段或字母　　<g1>和<g2>的值之间才符合该条件。

3）<f> NOTLIKE <g> [ESCAPE <h>]

该条件只能用于字符类型字段。要符合该条件，数据库字段　　<f> 中的值必须（不能）符合<g>的模式。在指定 <g>时，可使用下列两个通配符：

_（下划线）表示单个字符　%（百分号）表示任意字符串，包括空字符串

4）<f>　　....., <gn>)

要满足该条件，数据库字段　<f> 中的值必须（不能）等于括号内列表中的一个值。

在该变式中，在括号和比较字段　　<gi> 之间不能有空格，但在比较字段之间可以有空格。

5）<f> IS [NOT] NULL

数据库字段 <f> 中的值必须（不能）等于　　NULL 值。

6）<f> NOTIN <seltab>

要满足该条件，数据库字段　<f> 的值必须（不能）符合选择表　　<seltab> 中指定的条件。选择表是一个特殊的内表，报表用户可在选择屏幕上填充它。通常是使用　　SELECT-OPTIONS 或 RANGES来创建选择表，但是也可以按照创建和处理内表中的说明定义。

（2）使用逻辑链接操作符组合条件

可以使用逻辑链接操作符　　AND、OR 和 NOT 按照任意顺序来组合六个基本　　WHERE条件。

如果希望指定几个必须同时满足的条件，可按照下列方法用　　AND 组合它们：

....WHERE <condition1> AND <condition2> AND <condition3> AND...

如果需要指定几个条件，　至少要满足其中的一个条件，　就可按照下列方法用　OR 组合它们：

....WHERE <condition1> OR <condition2> OR <condition3> OR...

如果只希望选择那些不符合指定条件的表条目，就可用　　NOT 转化条件，如下所示：

....WHERE NOT <condition>

NOT的优先级比 AND 高，AND的优先级比 OR 高。但是，可使用括号来定义处理的顺序。这些括号都必须加空格。

9.2.5　指定行选择条件

（1）指定条件的列表

要在运行时指定一系列条件来选择一定数目的特定行，可在　　SELECT 语句中使用下列

WHERE子句的特殊变式：

SELECT.....FOR ALL ENTRIES IN <itab> WHERE <condition> .....

在条件 <condition> 中，可按前面的叙述将内部字段或字母指定为比较值。也可以将内表<itab> 的列或字母用作比较值。在 WHERE条件中，这些列将用作占位符。

该 SELECT 语句的结果集是 SELECT 语句的所有结果集的联合， 这些结果集是用 <itab> 中的相应值在每一行上替换占位符的结果，将从结果集中删除重复行。

如果使用了 WHERE子句的该变式，就不要使用 ORDER BY 子句。

（2）给行分组

要将数据库表的一组行中的内容组合到单个行中，可按如下格式使用 SELECT 语句的GROUP BY子句：

SELECT [DISTINCT] <a1><a2> ..

FROM clause INTO clause GROUP BY <f1><f2> ....

SELECT [DISTINCT] <a1><a2> ...

FROM clause INTO clause GROUP BY (<itab>)

该组由 <f1><f2> ..... 指定的列中包含有相同值的行组成。

（3）指定行的顺序

在读取行的集合时，可使用 SELECT 语句的 ORDERBY 子句指定这些行的顺序，按该顺序将这些行传送给 ABAP/4 程序。

如果不使用 ORDER BY 子句，将不定义选择的行的顺序。

1）可根据主码或明确指定的字段给这些行排序，可使用下列形式的 ORDER BY 子句：

SELECT * ..... ORDER BY PRIMARY KEY.

如果使用 ORDER BY PRIMARY KEY选项，系统将按主码以升序给所选行排序。

2）按指定的字段排序，可使用下列语法：

.. ORDER BY <f1> [ASCENDING|DESCENDING] <f2> [ASCENDING|DESCENDING] ...

按指定的表字段 <f1>, <f2>, ... 给所选行排序。可以在每个字段名后指定选项ASCENDING或 DESCENDING为每个表字段明确地指定排序顺序。

标准的排序顺序是升序。 如果指定了多个字段，那么系统首先按 <f1> 给所选行排序，然后是 <f2> ，依此类推。

9.3 添加数据

9.3.1 添加单一行

要在数据库表中添加一单行，可使用下列 INSERT 语句的任一变式。 INSERT 语句的基

本格式如下所示：

INSERT INTO <dbtab> [CLIENT SPECIFIED] VALUES <wa>.

工作区 <wa>中的内容将写进数据库表 <dbtab> 中。必须在程序中使用 TABLES 语句声明该数据库表。 工作区 <wa>的长度至少要等于数据库表的表工作区长度。 为了保证工作区具有与数据库表相同的结构，可通过 DATA 或 TYPES 语句用 LIKE <dbtab> 选项来定义它。

9.3.2 添加多行

要使用 INSERT 语句将几行从内表中添加到数据库表中，使用下列语法：

INSERT <dbtab> [CLIENT SPECIFIED] FROM TABLE <itab>

[ACCEPTING DUPLICATE KEYS].

如果需要在运行时间指定数据库表名，使用下列语法：

INSERT (<dbtabname>) [CLIENT SPECIFIED] FROM TABLE <itab>

[ACCEPTING DUPLICATE KEYS].

此语句在单个操作中将内表 <itab> 中的所有行添加到数据库表中。

9.4 更新数据

9.4.1 更新单一行

要使用 UPDATE 语句更改单行，可使用 UPDATE语句的短格式，带有 SET 子句并指定了完整的 WHERE条件的 UPDATE 语句。

UPDATE语句的短格式如下所示：

UPDATE <dbtab> [CLIENT SPECIFIED] FROM <wa>.

和

UPDATE <dbtab> [CLIENT SPECIFIED].

在第一个语句中， 工作区 <wa>中的内容将覆盖数据库表 <dbtab> 的行，该数据库表与 <wa> 具有相同的主码。必须在程序中用 TABLES 声明该数据库表。

在第二个语句中，没有指定工作区 <wa>。但表工作区 <dbtab> 中的内容将覆盖具有相同主码的数据库表的行。

9.4.2 更新多行

要使用 UPDATE 语句在数据库表中更改多行，使用下列语法：

UPDATE <dbtab> [CLIENT SPECIFIED] SET <S1> ..<Sn> [WHERE <condition>].

使用 WHERE子句选定要更改的行。 如果不指定 WHERE子句， 将更改所有的行。 必须在程序中用 TABLES 语句声明该数据库表。 CLIENT SPECIFIED 选项用于关闭自动集团处理。

9.4.3 使用内表更新多行

要通过 UPDATE 语句使用内表在数据库表中更改多行，使用下列语法：

UPDATE <dbtab> [CLIENT SPECIFIED] FROM TABLE <itab>.

如果需要在运行时间指定数据库表的名称，就要使用下列语法：

UPDATE (<dbtabname>) [CLIENT SPECIFIED] FROM TABLE <itab>.

内表 <itab> 中的行将覆盖数据库表中具有相同主码的行。

内表中行的长度至少要等于数据库表中行的长度。 为了保证内表的结构与数据库表的结构相同，就必须通过 DATA 或 TYPES 语句使用 LIKE <dbtab> 选项来定义。

如果因为不存在带有指定关键字的行而使系统不能更改行， 那么， 系统不会终止整个操作，而是继续处理内表中的下一行。

9.5 添加或更改数据

要在数据库表中插入一行， 而不考虑该行的主码是否已经存在， 使用 MODIFY 语句。 有两种可能：

如果数据库表中没有哪一行的主码与即将插入行的主码相同，那么， MODIFY的操作就与 INSERT 的操作相同，即添加该行。

如果数据库中已经包含主码与即将插入行的主码相同的行， 那么， MODIFY 的操作就与UPDATE的操作相同，即更改该行。

9.5.1 插入单行

要插入单行，使用下列语法：

MODIFY <dbtab> [CLIENT SPECIFIED] [FROM <wa>].

如果需要在运行时间指定数据库表的名称，使用下列语法：

MODIFY (<dbtabname>) [CLIENT SPECIFIED] [FROM <wa>].

9.5.2 插入多行

要插入多行，使用下列语法：

MODIFY <dbtab> [CLIENT SPECIFIED] FROM TABLE <itab>.

如果需要在运行时间指定数据库表的名称，使用下列语法：

MODIFY (<dbtabname>) [CLIENT SPECIFIED] FROM TABLE <itab>.

9.6 删除数据

9.6.1 删除单行

要使用 DELETE 语句删除单行，可以将 DELETE 语句与完整的 WHERE条件一起使用。使用 DELETE 语句的短格式。

DELETE <dbtab> [CLIENT SPECIFIED] FROM <wa>.

DELETE <dbtab> [CLIENT SPECIFIED].

在第一个语句中，将从数据库表 <dbtab> 中删除主码与 <wa>中指定的主码相同的行。必须在程序中用 TABLES 语句声明该数据库表。

在第二个语句中， 没有指定工作区 <wa>。而是从数据库表中删除主码与表工作区 <dbtab> 中指定的主码相同的行。

9.6.2 删除多行

要在单个操作中从数据库表 <dbtab> 中删除多行，可使用如下格式的 DELETE 语句：

DELETE FROM <dbtab> [CLIENT SPECIFIED] WHERE <conditions>.

如果需要在运行时间指定数据库表的名称，使用下列语法：

DELETE FROM (<dbtabname>) [CLIENT SPECIFIED] WHERE <conditions>.

9.6.3 使用内表删除多行

要通过 DELETE 语句用内表从数据库表中删除多行，使用下列语法：

DELETE <dbtab> [CLIENT SPECIFIED] FROM TABLE <itab>.

如果需要在运行时间指定数据库表的名称，使用下列语法：

DELETE (<dbtabname>) [CLIENT SPECIFIED] FROM TABLE <itab>.

这些语句从数据库表中删除主码与内表 <itab> 中的某行相同的行。 内表的长度至少与数据库表的主码的长度相同。

如果因为数据库表中没有哪一行的主码与指定的相同， 从而系统没有删除任何行， 那么，系统不会终止整个操作，而是继续处理内表的下一行。

如果已经处理了内表中所有的行， SY-SUBRC 将设置为 0 。否则，设置为 4 。在稍后的时候， 从内表的总行数中减去 SY-DBCNT 中给出的实际已删除的行数， 就可计算系统尚未删除的行数。如果内表是空的，那么 SY-SUBRC 和 SY-DBCNT 都将设置为 0 。

9.7 使用光标读取数据

9.7.1 打开光标

使用光标几乎可从任何 SELECT 语句的结果集中获得下一行（或一些行） 。为此，按如下格式使用 OPEN CURSOR 语句将光标与相关的 SELECT 语句链接起来：

OPEN CURSOR [WITH HOLD] <c> FOR SELECT .....
[WHERE <conditions>].

首先必须将光标 <c>定义为类型 CURSOR，如果使用 WITH HOLD选项，那么在自身的 SQL 数据库提交之后，该光标将保留打开状态。

9.7.2 用光标读取数据

在打开光标后，就可使用 FETCH 语句从 OPENCURSOR 语句生成的结果集中读取下一行，如下所示：

FETCH NEXT CURSOR <c> INTO <target>.

所选行都将读到 INTO 子句指定的目标区中。

9.7.3 关闭光标

必须使用 CLOSE CURSOR 语句关闭不再需要的光标，如下所示：

CLOSE CURSOR <c>.

下列情况将自动关闭光标：

1）执行 COMMIT WORK 或 ROLLBACK WORK 语句时；

2）当执行自身的 SQL 数据库提交或取消时；

3）更改屏幕时；

4）执行远程功能调用时。

如果在 OPEN CURSOR 语句中使用 WITH HOLD 选项，自身的 SQL 中的数据库提交将不会关闭光标。

9.8 数据簇

9.8.1 向内存中存储数据对象

要将数据对象从 ABAP/4 程序写入 ABAP/4 内存，使用下列语句：

EXPORT <f1> [FROM <g1>] <f2> [FROM <g2>] ... TO MEMORY ID <key>.

此语句将列表中指定的数据对象存储为 ABAP/4 内存中的数据簇。如果忽略选项 FROM <gi>，则将数据对象 <fi> 存储到自己的名称之下。如果使用该选项，则将数据对象 <gi> 存储到<fi> 下面。ID <key> 用于标识内存数据，不得超过 32 个字符。

EXPORT语句总是完全改写 ID <key>相同的任何现有数据簇的内容。对于有表头行的内表，只可以存储表本身，而不能存储表头行。

9.8.2 从内存中读取数据对象

要将 ABAP/4 内存中的数据对象读到 ABAP/4 程序中，使用下列语句：

IMPORT <f1> [TO <g1>] <f2> [TO <g2>] ... FROM MEMORY ID <key>.

此语句从 ABAP/4 内存的数据簇读取列表中指定的数据对象。如果忽略选项 TO <gi>，则将内存中的数据对象 <fi> 赋给程序中的同名数据对象。如果使用此选项，则将内存中的数据对象 <fi> 写入字段 <gi> 中。ID <key> 用于标识内存数据，不得超过 32 个字符。

此语句不进行这种检查：即内存中的对象结构与要写入的结构是否匹配。因为数据是按位进行传送的，所以不匹配的结构可能会引起不一致。

9.8.3 删除内存中的数据簇

要删除 ABAP/4 内存中的数据对象，使用下列语句：

FREE MEMORY [ID <key>].

如果不附加 ID <key>，则此语句删除整个内存，包括此前用 EXPORT存储到 ABAP/4 内存中的所有数据簇。附加 ID <key> 之后，该语句只删除用此名称命名的数据簇。

因为删除整个内存会导致任何系统例程内存内容的丢失，所以应只使用附加有 ID 的FREE MEMOR语句。

十文件

10.1 打开文件

要在应用服务器上打开文件，使用如下 OPEN DATASET语句：

OPEN DATASET <dsn> [options].

此语句打开文件 <dsn>。如果不指定任何模式选项，则文件将按二进制模式打开。如果系统不能打开文件，则将系统字段 SY-SUBRC 设置为 0，否则 SY-SUBRC 返回 8。

可以将文件名 <dsn>指定为字母或包含文件名的字段。如果未指定路径，则系统将在应用服务器上 SAP 系统运行的目录中打开文件。要打开文件，运行 SAP 系统的用户必须在操作系统级有相应的权限。

10.1.1 接受操作系统消息

尝试打开文件后，要接受操作系统消息，使用如下 OPEN DATASET语句的 MESSAGE选项：

OPEN DATASET <dsn> MESSAGE <msg>.

系统将在变量 <msg>中放置相关的操作系统消息。

10.1.2 打开文件读取

要打开文件进行读访问，使用如下 OPEN DATASET语句的 FOR INPUT 选项：

OPEN DATASET <dsn> FOR INPUT.

此语句打开文件用于读取。文件必须已经存在，否则系统将 SY-SUBRC 设置为 8 并且忽略此命令。如果文件已打开（可能用于读、写或追加），系统将复位到文件的起始位置。但是在重新打开文件之前使用 CLOSE 语句是良好的编程风格。

10.1.3 打开文件写入

要打开文件进行写访问，使用如下 OPEN DATASET语句的 FOR OUTPUT选项：

OPEN DATASET <dsn> FOR OUTPUT.

此语句打开文件用于写入。 如果文件不存在， 则创建文件。 如果文件已存在但处于关闭状态，则删除其内容。如果文件已存在且已打开（可能为读、写或追加） ，则复位到文件的起始位置。如果系统可以打开文件，则 SY-SUBRC 设置为 0，否则 SY-SUBRC 返回 8。

10.1.4 打开文件追加

要打开文件追加数据，使用 OPEN DATASET语句的 FOR APPENDING选项：

```
OPEN DATASET <dsn> FOR APPENDING.
```

此语句打开文件在文件末尾写入数据。 如果文件不存在， 则创建文件。 如果文件已存在但处于关闭状态， 系统将打开文件并定位到文件末尾。 如果文件已存在且已打开 （可能为读、写或追加），将定位设置到文件末尾。 SY-SUBRC总是返回 0。

10.1.5 指定二进制模式

要用二进制模式处理文件，使用 OPEN DATASET语句中的 IN BINARY MODE 选项：

```
OPEN DATASET <dsn> IN BINARY MODE [FOR ....].
```

如果从以二进制模式打开的文件中读取数据或向此类文件中写入数据， 系统将逐字节地传输数据。 在传输期间， 系统不解释文件内容。 在将文件内容写入到另一文件时， 系统将传输源文件的所有字节。在从文件读取数据到字段时，传输的字节数目取决于目标字段大小。

在读取之时或之后， 可以用其它 ABAP/4 语句给目标字段定址， 系统将根据数据类型解释字段内容。

10.1.6 指定文本模式

要用文本模式处理文件，使用 OPEN DATASET语句的 TEXT MODE选项：

```
OPEN DATASET <dsn>FOR ....IN TEXT MODE.
```

如果从以文本模式打开的文件中读取数据或向此类文件中写入数据，数据将逐行传输。系统假定文件为行结构。

10.1.7 在指定位置打开文件

要在指定位置打开文件，使用 OPEN DATASET语句的 AT POSITION 选项：

```
OPEN DATASET <dsn> [FOR ....] [IN ... MODE] AT POSITION <pos>.
```

此语句打开文件 <dsn>并为读写数据定位位置 <pos>, 该位置从文件起始处起按字节计算。在文件起始处以前，不能指定位置。以二进制模式工作时，指定位置 <pos>相当有用，因为文本文件的物理表示法取决于操作系统。

10.2 关闭文件

要在应用服务器上关闭文件，使用 CLOSE DATASET语句：

```
CLOSE DATASET <dsn>.
```

此语句关闭文件 <dsn>，文件命名在打开文件中有说明。

只有在下次为写入打开文件期间要删除文件内容时，关闭文件才有必要。为避免错误并使程序易于阅读，在使用下一条 OPEN DATASET语句之前应关闭文件。通过 CLOSE 语句，可将文件分成逻辑块并易于维护。

10.3 删除文件

要在应用服务器上删除文件，使用 DELETE DATASET语句：

DELETE DATASET <dsn>.

此语句删除文件 <dsn>。文件命名在打开文件中有说明。如果系统可以删除文件 <dsn>，则 SY-SUBRC 返回 0，否则 SY-SUBRC返回 4。

10.4 向文件中写入数据

要向在应用服务器上的文件写入数据，使用 TRANSFER语句：

TRANSFER <f> to <dsn> [LENGTH <len>].

此语句将字段 <f> 的值写入文件 <dsn>，可以用 OPENDATASET语句指定传输模式。如果不能打开文件以写入，则系统将尝试用二进制模式打开文件，或为此文件使用 OPENDATASET语句的选项。

字段 <f> 的数据类型可以是基本型，或者是不包含作为组件的内表格的字段字符串（内表数据需通过结构进行写入）。内表格不能在一次执行中写入文件。通过 LENGHT 选项，可以指定传输数据的长度 <len>。系统将第一个 <len> 字节写入文件。如果 <len> 太小，则截断超出的字节。如果 <len> 太大，系统将在传输行的右端填入空格。

10.5 从文件中读取数据

要从应用服务器上的文件中读取数据，使用 READ DATASET语句：

READ DATASET <dsn> INTO <f> [LENGTH <len>].

此语句从文件 <dsn>中读取数据赋值给变量 <f>。要决定把从文件中读取的数据赋值给哪个变量，必须清楚文件结构。

可以用 OPENDATASET语句指定传输模式。如果没有打开文件以读取，则系统将尝试以二进制模式打开文件或为此文件使用 OPEN DATASET语句的选项。在读取操作成功后，SY-SUBRC将返回 0。当到文件末尾时，SY-SUBRC将返回 4。当不能打开文件时，SY-SUBRC将返回 8。

如果以二进制模式工作，则可以使用 LENGTH 选项查找实际传输给字段 <f> 的数据长度。系统可用变量 <len> 值设置该长度。

10.6 使用服务器上的文件

（1）通过用户对话向服务器写入数据

CALL FUNCTION 'DOWNLOAD'

（2）不通过用户对话向服务器写入数据

CALL FUNCTION 'WS_DOWNLOAD'

（3）通过用户对话从服务器读取数据

CALL FUNCTION 'UPLOAD'

（4）不通过用户对话从服务器读取数据

CALL FUNCTION 'WS_UPLOAD'

（5）检查服务器上的文件

CALL FUNCTION 'WS_QUERY'

十一报表

11.1 PARAMATERS为变量定义输入字段

可以按照用 DATA 语句声明字段的方式用 PARAMETERS 语句声明字段。用 PARAMETERS 语句声明的字段称为参数。 通常，输入字段的所有参数都将出现在选择屏幕上。 系统处理输入屏幕时，报表用户在这些输入字段中键入的值将被分配给相应的参数。

要声明参数及其数据类型，使用 PARAMETERS 语句，如下所示：

PARAMETERS <p>[(<length>)] <type> [<decimals>].

该语句创建参数 <p>。附加项 <length> 、<type> 和 <decimals> 与 DATA 语句相同。用户启动报表程序时， <p>的输入字段将出现在选择屏幕中。

11.1.1 给参数分配缺省值

要为将显示在选择屏幕上的输入字段分配缺省值，使用 PARAMETERS 语句的 DEFAULT 选项。语法如下：

PARAMETERS <p> ..... DEFAULT <f> .....

<f> 可以是文字或字段名。如果指定字段名，则系统将按缺省值处理该字段的内容，报表用户可以在选择屏幕中更改缺省值。

系统在初始化事件 INITIALIZATION 之前将缺省值传输给参数。因此，对于那些在用户启动程序时已经被填充的字段，应该使用字段名而不是文字作为字段的缺省值。

11.1.2 禁止参数显示

要禁止在选择屏幕中显示参数， 使用 PARAMETERS 语句的 NO-DISPLAY 选项。语法如下：

PARAMETERS <p> ..... NO-DISPLAY .....

11.1.3 允许参数接收大小写

要允许用户用大写或小写字母输入参数值，使用 PARAMETERS 语句的 LOWER CASE 选项。语法如下：

```
PARAMETERS <p> ..... LOWER CASE .....
```

如果没有 LOWER CASE 选项，系统将所有输入值更改为大写。如果使用 LIKE 选项从 ABAP/4 词典中引用字段，参数将接受 ABAP/4 字段的所有属性，不能更改这些属性，也不能使用 LOWER CASE 选项，必须在 ABAP/4 词典中定义是否可以输入大写或小写值。

11.1.4 必须输入的参数字段

要保证参数必须输入，使用 PARAMETERS 语句的 OBLIGATORY 选项。语法如下：

```
PARAMETERS <p> .....OBLIGATORY .....
```

使用该选项时，用户如果不在选择屏幕的这个字段上输入值，程序就无法继续执行。

11.1.5 创建复选框

要为参数输入定义复选框，使用 PARAMETERS 语句的 AS CHECKBOX 选项。语法如下：

```
PARAMETERS <p> ..... AS CHECKBOX .....
```

参数 <p> 按长度为 1 的类型 C 创建。在这种情况下，不允许使用附加选项 TYPE 和 LIKE。<p> 的有效值是 ‘ ’ 或 ‘ X’，这些值在用户单击选择屏幕上的复选框时赋给参数。

11.1.6 创建单选按钮组

要为参数输入定义单选按钮组，使用 PARAMETERS 语句的 RADIOBUTTON GROUP 选项。语法如下：

```
PARAMETERS <p> ..... RADIOBUTTON GROUP <radi>.....
```

参数 <p> 按类型 C，长度 1 创建，并分配到组 <radi> 。字符串 <radi> 的最大长度是 4 。允许使用附加的 LIKE 选项，但是必须参阅类型为 C、长度为 1 的字段。必须为每个 <radi> 组分配至少两个参数，每个组中只有一个参数可以用 DEFAULT 选项分配的缺省值，该值必须是 ‘ X ’。

当用户单击选择屏幕中的单选按钮时，相应的参数被激活 （分配值 ‘ X ’），同时同组的其它参数被设为非活动的（赋值 ‘ ’ ）。

11.1.7 使用内存缺省值

PARAMETERS 语句的 MEMORY-ID 选项允许从全局 SAP 内存使用缺省值。语法如下：

```
PARAMETERS <p> ..... MEMORY ID <pid>.....
```

使用该选项时，以名称 <pid> 存储在全局用户相关的 SAP 内存中的值将作为 <p> 的缺省值出现在选择屏幕上。 <pid> 最长 3 个字符，并且不能用引号封闭。

可以使用全局 SAP 内存在程序间传递保留在事务限制以外的值。用户在整个终端进程期间都可利用该内存，并且所有并行进程使用相同的内存。因此 SAP 内存包含的内容要比局限于事务的 ABAP/4 内存更广泛。参数可以按用户在名为 <pid> 的用户主记录中的说明 进行设置。

11.1.8 分配匹配的代码对象

要给参数分配匹配代码对象，使用 PARAMETERS 语句的 MATCHCODE OBJECT 选项。语法如下：

```
PARAMETERS <p> ..... MATCHCODE OBJECT <obj> .....
```

匹配代码对象 <obj> 的名称必须是 4 字符的变量名，并且不能用引号封闭。如果使用该选项，则可能的条目按钮将在参数 <p> 的输入字段之后出现。用户按下该按钮时，其结果是为执行输入字段的匹配代码选择。

11.1.9 分配修改组

要将参数分配给修改组，使用 PARAMTERS 语句的 MODIF ID 选项，如下所示：

```
PARAMETERS <p> ..... MODIF ID <key> .....
```

修改组 <key> 的名称必须是不带引号 的 3 字符变量名。MODIF ID 选项总是把 <key> 分配到内表 SCREEN 的 SCREEN-GROUP1 列。

在 AT SELECTION-SCREEN OUTPUT 事件中，分配给修改组的参数可以用 LOOP AT SCREEN/MODIFY SCREEN 语句按整个组进行处理 。

11.2 SELECT-OPTIONS 定义选择标准

系统为每个 SELECT-OPTIONS 语句创建选择表。选择表的目的是按标准化的方式保存复合选择限制。选择表是一个带表头行的内表。它的行结构是字段字符串，由四个组件构成，即 SIGN、OPTION、LOW 和 HIGH。

SIGN 表示保存在 OPTION 中的运算符是否需要翻转， 允许值 是 I 和 E；OPTION 包含选择运算符； LOW 和 HIGH 中的内容为间隔选择指定上界和下界。

要创建在选择屏幕中填充的选择标准，可以使用 SELECT-OPTIONS 语句，如下所示：

```
SELECT-OPTIONS <seltab> FOR <f>.
```

该语句创建选择表 <seltab> ，该表与数据库表的列 <f> 或内部字段 <f> 相连接。该 数据库表必须在程序中用 TABLES 语句声明。选择表 <seltab> 由报表用户在选择屏幕中填充。

11.2.1 给选择标准分配缺省值

要给将显示在选择屏幕的选择标准分配缺省值， 使用 SELECT-OPTIONS 语句的 DEFAULT 选项。语法如下：

SELECT-OPTIONS <seltab> FOR <f>

DEFAULT <g> [TO <h>]

[OPTION <op>]

SIGN <s>.

<g>和<h>的缺省值可以是实际值 （在单引号中）或是字段名，该字段的值将用做缺省值。<g>用于设定 LOW字段， <h>用于设定 HIGH字段。

对于单值选择， <op>可以是 EQ、NE、GE、GT、LE、LT、CP 或 NP，缺省值是 EQ。对于期间选择， <op>可以是 BT 或 NB，缺省值是 BT 。

SIGN <s>可以是 I （包括）和 E（排除），缺省值是 I 。

11.2.2 限制为单行

要把用户对选择表的访问限制在单行， 使用 SELECT-OPTIONS语句的 NO-EXTENSION选项，语法如下所示：

SELECT-OPTIONS <seltab> FOR <f> ..... NO-EXTENSION .....

如果指定该选项， 则选择屏幕上不出现右箭头， 并且用户不能访问 “ 复杂选择 ” 窗口。

11.2.3 限制为单值

要将选择屏幕上选择标准的外观限制为单值选择，使用 SELECT-OPTIONS语句的 NO INTERVALS选项。语法如下所示：

SELECT-OPTIONS <seltab> FOR <f> ..... NO INTERVALS .....

如果指定该选项，则 “ 到 ” 字段不出现在选择屏幕上，选择屏幕上的输入将限制为单值选择。但是，用 户可以在 “ 复杂选择 ” 屏幕中输入间隔选择。

11.3 空行、下划线和注释

11.3.1 空行

要在选择屏幕上产生空行，使用 SELECTION-SCREEN语句的 SKIP 选项。语法如下：

SELECTION-SCREEN SKIP [<n>].

该语句产生 <n>个空行，其中 <n>的值可以是 1 到 9 。要产生单个空行，可以省略 <n>。

11.3.2 下划线

要在选择屏幕中给一行或行中的一部分加下划线，使用 SELECTION-SCREEN语句的 ULINE 选项。语法如下：

SELECTION-SCREEN ULINE [[/]<pos(len)>] [MODIF ID <key>].

该语句创建下划线。

如果不使用格式 选项 <pos(len)> ，则 在当前行下 面创建新行 。如果使用 格式选项

<pos(len)> ，下划线将从当前行的位置 <pos>处开始，连续 <len> 个字符。通过一行中的 几个元素，也可以不通过 <pos> 来指定 (<len>) 。可以使用可选的斜杠 (/) 请求换行。

11.3.3 注释

要在选择屏幕中书写文本， 使用 SELECTION-SCREEN 语句的 COMMENT 选项。语法如下：

```
SELECTION-SCREEN COMMENT [/]<pos(len)><name> [FOR FIELD <f>]
[MODIF ID <key>].
```

使用该选项时必须定义格式（开始位置和长度） 。对于 <name>，可以指定文本符号或指定最大长度为 8 个字符的字段名。该字符字段不能用 DATA 语句声明，而是通常自动地按长度 <len> 生成。必须在 INITIALIZATION 事件中动态地填充该字符字段。

将显示文本 <name>，从列 <pos> 开始， 长度为 <len> 。如果不使用斜杠 (/) ，注释将写到当前行，否则将创建新行 。

要把文本标签分配给参数或选择选项，使用 FOR FIELD <f> 选项。 <f> 可以是参数或选择标准的名称。因此，如果用户在选择屏幕的注释上请求帮助，则显示分配给字段 <f> 的帮助文本。

选择屏幕注释的 MODIF ID <key> 选项与为 PARAMETERS 语句所描述的相同。

11.4 将几个元素放在同一行

要将参数或注释集合定位在选择屏幕的一行上， 必须在由下列两条语句封闭的块中声明元素：

```
SELECTION-SCREEN BEGIN OF LINE.
...
SELECTION-SCREEN END OF LINE.
```

使用该选项时不会显示选择文本，要显示选择文本，必须通过与 COMMENT 选项一起使用 SELECTION-SCREEN 语句提供描述。另外，不要把斜杠与格式选项 <pos(len)> 一起使用。在该格式选项中， 可以在上述语句中省略位置说明 <pos>。 随后对象将定位在行中的当前位置。

11.5 创建元素块

要在选择屏幕上创建元素逻辑块，使用 SELECTION-SCREEN 语句的 BEGIN OF BLOCK 选项选择块的开始，然后定义各个元素并用 END OF BLOCK 选项选择块的结束，如下所示：

```
SELECTION-SCREEN BEGIN OF BLOCK <block>
                [WITH FRAME [TITLE <title>]]
                [NO INTERVALS].
```

...

SELECTION-SCREEN END OF BLOCK <block>.

必须为每个块定义名称 <block> ，可以嵌套块。

如果增加 WITH FRAME 选项，在块的周围将画外框。最多可以嵌套 5 层带外框的不同块。可以使用 TITLE 选项给每个外框增加标题， <title> 可以是文本符号或字符文字。 该字符字段的长度是外框的 宽度，该宽度自动按照外框的嵌套深度进行设置。

11.6 在应用工具条上创建按钮

在选择屏幕的应用工具条中最多可以创建 5 个按钮，这些按钮自动与功能键相连接。语法如下所示：

SELECTION-SCREEN FUNCTION KEY <i>.

<i> 必须在 1 到 5 之间。 必须在 ABAP/4 词典字段 SSCRFIELDS-FUNCTXT_0<i>中指定要在运行时间出现在按钮上的文本。

必须用 TABLES 语句声明 SSCRFIELDS。 当 用户单击该按钮时， FC0<i> 输入到字段 SSCRFIELDS-UCOMM中，可以在事件 AT SELECTION-SCREEN中检查该字段。

11.7 在选择屏幕上创建按钮

要在选择屏幕上创建按钮， 可以与 PUSHBUTTON参数一起使用 SELECTION-SCREEN语句。语法如下：

SELECTION SCREEN PUSHBUTTON [/]<pos(len)><name>

USER-COMMAND <ucom> [MODIF ID <key>].

在 <name>中指定的文本是按钮文本。 对于 <ucom>， 必须指定最多为 4 字符的代码。 当用户在选择屏幕上单击按钮时， <ucom>被输入词典字段 SSCRFIELDS-UCOMM。

必须使用 TABLES 语句声明 SSCRFIELDS。可以在事件 AT SELECTION-SCREEN中检查 SSCRFIELDS-UCOMM字段的内容 。

十二事件

事件关键字	事件
INITIALIZATION	在显示选择屏幕之前的点
AT SELECTION-SCREEN	选择屏幕仍然活动时，处理用户在选择屏幕上输入之后的点
START-OF-SELECTION	处理选择屏幕之后的点
GET <table>	点位于逻辑数据库提供数据库表 <table> 的行处
GET <table> LATE	处理所有表之后的点，在逻辑数据库的结构中，使这些表位于数据库表 <table> 的下层

END-OF-SELECTION	处理完逻辑数据库提供的所有行之后的点
TOP-OF-PAGE	启动新页时，列表处理中的点
END-OF-PAGE	结束页时，列表处理中的点
AT LINE-SELECTION	用户在该点处选择行
AT USER-COMMAND	用户在该点处按下功能键或在命令字段中输入命令
AT PF<nn>	用户在该点处按下有功能代码　　PF<n>的功能键

12.1 INITIALZATION

启动已定义选择屏幕的程序时，　　系统通常首先处理此选择屏幕。　　如果希望在处理选择屏幕之前执行过程块，可以把它分配到事件关键字　　INITIALIZATION　。

在此块中，　　指定初始化选择屏幕的语句。

12.2 AT SELECTION-SCREEN

12.2.1　处理特殊输入字段

要在处理选择屏幕的特殊输入字段之后启动过程块，按照下列方法使用关键字　　AT SELECTION屏幕：

AT SELECTION-SCREEN ON <field>.

系统处理了变量　<field>　的输入字段之后，启动相应过程块。如果从此过程块中发送ERROR MESSAGE，则系统再次显示选择屏幕，用户必须更改变量　　<field>　的输入字段。

12.2. 处理多重选择

将特殊选择标准的复杂选择输入到选择屏幕的　　“多重选择　”窗口，并处理此窗口之后，可以调用过程块，按照下列方法使用　　AT SELECTION-SCREEN语句：

AT SELECTION-SCREEN ON END OF <seltab>.

在处理选择标准　<seltab>　的“复杂选择”窗口结束处启动相应过程块。　　可以使用这些选项检查内表　<seltab>　中的条目。

12.2.3　创建输入值列表

通过按照下列方法使用　AT SELECTION-SCREEN语句，可以为选择屏幕上的输入字段创建可能输入值的列表：

AT SELECTION-SCREEN ON VALUE-REQUEST FOR <field>.

如果使用此语句，　在选择屏幕上选定参数或选择标准　　<field>　的输入字段时，　会自动　紧跟着该字段出现可能的登录按钮　　。

只可以在报表程序中使用此语句　　。在逻辑数据库程序中，可以使用　　PARAMETERS和SELECT-OPTIONS语句的　VALUE-REQUEST选项。必须在　AT SELECTION-SCREEN ON VALUE REQUEST语句的过程块内为　<field>　编写建议值列表程序。

12.2.4　创建输入字段帮助

通过按照下列方法使用 AT SELECTION-SCREEN 语句，可以为选择屏幕上的输入字段创建帮助：

AT SELECTION-SCREEN ON HELP-REQUEST FOR <field>.

如果使用此语句，用户在选择屏幕上选择 <field> 的输入字段，并按下 F1 键时，将显示帮助文本。

只可以在报表程序中使用此语句。在逻辑数据库程序中，可以使用 PARAMETERS 和 SELECT-OPTIONS 语句的 HELP-REQUEST 选项。必须在 AT SELECTION-SCREEN ON HELP REQUEST 语句的过程块内编写帮助文本的程序。

12.2.5 处理单选按钮组

在选择屏幕上处理完单选按钮组之后，要启动过程块，按照下列方法使用关键字 AT SELECTION 屏幕：

AT SELECTION-SCREEN ON RADIOBUTTON GROUP <radi>.

系统处理单选按钮组 <radi> 后启动相应过程块。如果从过程块中发送 ERROR MESSAGE，则系统再次显示选择屏幕，用户必须更改单选按钮 <radi> 的输入字段。

12.2.6 处理输入字段块

在选择屏幕上处理完元素块之后启动过程块，按照下列方法使用关键字 AT SELECTION 屏幕：

AT SELECTION-SCREEN ON BLOCK <block>.

系统处理完元素块 <block> 时启动相应过程块。如果从此过程块中发送 ERROR MESSAGE，则系统再次显示选择屏幕，用户必须更改块 <block> 的输入字段。

12.2.7 选择屏幕的 PBO

要在每个 ENTER 的选择屏幕 PBO 中启动过程块，使用下列 AT SELECTION-SCREEN 语句：

AT SELECTION-SCREEN OUTPUT.

12.3 START-OF-SELECTION

可能的事件过程块。例如，可以使用这些过程块设置内表的值，或将信息语句写到输出屏幕上。

在 START-OF-SELECTION 事件处，也处理没有附加到事件关键字的所有语句，或在 FORM-ENDFORM 块后写这些语句。

12.4 GET<table>

对于有附加逻辑数据库的报表程序，最重要的事件是逻辑数据库程序从数据库表中读取

了行的时刻。要在此事件处启动过程块，按照下列方法使用 GET 语句：

GET <table> [FIELDS <list>].

此语句之后，可以使用数据库表 <table> 的当前行。在表工作区 <table> 中提供了数 据。

（1）指定数据表的字段

要指定在 GET 事件处使用数据库表的哪一个字段，按照下列方法使用 GET 语句的 FIELDS 选项：

GET <table> [LATE] FIELDS <f1><f2> ...

使用 FIELDS 选项，从数据库表<table> 中，指定逻辑数据库程序只读取字段 <f1><f2> ... 和关键字段。 FIELDS 选项的使用可以引起相应性能改善。

12.5 GET<table> LATE

要在系统处理完逻辑数据库的所有数据库表之后启动过程块, 按照下列方法使用事件关键字 GET：

GET <table> LATE [FIELDS <list>].

与只使用 SELECT 语句的报表程序相似，在数据库表 <table> 的 SELECT 循环中， GET <table> LATE 语句的过程块直接出现在 ENDSELECT语句的前面 。FIELDS 选项的作用与 GET <table> 事件相同,

12.6 END-OF-SELECTION

要在系统读取和处理完逻辑数据库的所有数据库表之后定 义过程块，使用关键字 END-OF-SELECTION。

12.7 终止过程块

12.7.1 无条件的离开过程块

（1）转向 END-OF-SELECTION

可以立即离开任何过程块, 并通过按照下列方法使用 STOP语句转向 END-OF-SELECTION 过程块：

STOP.

运行 STOP 语句后，系统执行 END-OF-SELECTION过程块，并停止运行程序。

（2）转向输出屏幕

可以立即离开任何过程块（除在 AT 事件以外），并通过按照下列方法使用 EXIT 语句转向输出屏幕：

EXIT.

EXIT 语句后，系统显示输出列表，并停止运行程序。它不执行 END-OF-SELECTION过

程块。

（3）离开 AT 事件

如果在 AT 事件的过程块中使用 EXIT 语句（有以 AT 开头的事件关键字的所有事件），则系统离开此过程块，并转向下一个出现事件的过程块。

12.7.2 有条件的离开过程块

通过按照下列方法使用 CHECK 语句的两个变式，可以有条件地离开任何过程块：

CHECK <condition>.

如果 CHECK 语句中的条件为假，则系统离开过程块并转向下一个出现事件的过程块。

CHECK <seltab>.

如果数据库表的表工作区的内容与选择表 <seltab> 中的条件不匹配，并且将选择表 <seltab> 附加到该数据库表，则系统离开过程块。

12.7.3 无条件离开 GET事件

（1）转向当前数据表的下一行

要离开 GET 语句的过程块，并转向逻辑数据库同一层上的下一 GET 事件，按照下列方法 使用 REJECT：

REJECT.

此语句后，系统立即处理相同数据表的下一 GET 事件。这意味着它从当前表中启动相同过程块的新行。

（2）转向上级数据表的下一行

要离开 GET 语句的过程块，并执行逻辑数据库高层的下一 GET 事件，按照下列方法使用 REJECT 语句：

REJECT <dbtab>.

执行此语句后，系统将立即处理数据库表 <dbtab> 的下一 GET 事件。在链接到当前数 据库表的分层结构中，相比之下，数据库表 <dbtab> 必须位于高层。

12.7.4 有条件离开 GET事件

要有条件地离开 GET 事件，可以使用 CHECK 语句。用 CHECK 语句离开 GET 事件之后，系统将执行逻辑数据库相同层上的下一 GET 事件，它读取当前表的下一行。没有处理在逻辑数据库的分层顺序中较靠后的数据库表。

在 GET 事件中，可以使用 CHECK 语句的两种变式：

CHECK <condition>.

如果 CHECK 语句中的条件为假，则系统离开过程块。

CHECK SELECT-OPTIONS.

如果从当前数据库表中读取的行与所有选择表中的条件不匹配，并且将这些选择表连接到该数据库表，则系统离开过程块。

十三摘录数据集

由于内表具有固定的行结构，所以它不适于处理具有变化结构的数据集。出于这种考虑，ABAP/4 提供了创建所谓的摘录数据集的可能性。

摘录数据集是报表存储区中的顺序数据集，这意味着只能在循环中访问它的数据，不能像对内表操作那样通过索引访问其单个行。在报表运行过程中，系统可以正确地创建一个摘录数据集。至于内表，因为系统可根据需要展开它，所以原则上摘录数据集的大小是没有限制的。

摘录数据集由一系列预定义的结构的记录组成。但是，不是所有记录的结构都必须相同。在一个摘录数据集中，可以一个接一个地存储不同长度和结构的记录。不必为要存储的不同结构创建单个的数据集。这实际上大大减少了维护工作。

与内表不同，系统在存储摘录数据集时，将部分压缩摘录数据集。这减少了所需的存储空间。另外，不必在报表的开始处指定摘录数据集的结构，而可以在程序流过程中动态决定其结构。

13.1 创建并填充摘录数据集

13.1.1 定义摘录数据集为字段组

摘录数据集由一系列记录组成，这些记录可以有不同的结构。所有具有相同结构的记录形成一个记录类型。必须使用 FIELD-GROUPS 语句将摘录数据集的每个记录类型定义为字段组。

FIELD-GROUPS <fg>.

该语句定义了字段组 <fg>，字段组将几个字段组合到一个名称下。出于可读性的原因，最好在报表的开始处，即声明段之后定义字段组。

字段组不为字段保留存储空间，但它包含现有字段的指针。用记录填充摘录数据集时，这些指针将决定存储记录的内容。

可以定义特殊字段组 HEADER：

FIELD-GROUPS HEADER.

填充摘录数据集时，系统将自动用该字段组给所有其它字段组加上前缀。这意味着在摘录数据集中，字段组 <fg> 的记录总是首先包含 HEADER 字段组的字段。在给摘录数据集排序

时，系统将使用这些字段作为缺省的排序关键字。

13.1.2 给字段组分配字段

要确定将哪个字段包含进字段组中，使用 INSERT 语句：

INSERT <f1> ... <fn> INTO <fg>.

该语句定义字段组 <fg> 的字段。在给字段组分配字段之前，必须用 FIELD-GROUPS 语句定义字段组 <fg>。而对于字段 <fi>，就只能使用全局数据对象。不能将在子程序或功能模块中定义的局数据对象分配给字段组。

INSERT 语句与 FIELD-GROUPS 语句一样，既不保留存储空间，也不转换值。执行报表时，可给字段组分配字段，直到首次使用该字段组填充摘录记录。从这时起，记录的结构就已固定，不再改变。简而言之，如果还没有使用字段组，就可以动态地给它分配任意值。

由于特殊字段组 HEADER是每个摘录记录的一部分，所以在填充了第一个摘录记录后，就不能再更改该字段组。不必用字段填充字段组，如果定义 HEADER字段组，则只用 HEADER 中的字段填充摘录记录。

13.1.3 创建摘录数据集

要创建实际的摘录数据集，使用 EXTRACT 语句：

EXTRACT <fg>.

系统用报表的第一个 EXTRACT 语句创建摘录数据集并添加第一个摘录记录。并且每使用一个后续的 EXTRACT 语句，就向摘录数据集填充另一个摘取记录。

如果已经指定，则每个摘录记录都包含字段组 HEADER中的字段，并用它作为排序关键字，其后紧跟那些包含在字段组 <fg> 中的字段。在摘录过程中，系统用相应字段中的当前内容填充摘录记录。

13.2 处理摘录数据集

13.2.1 读取摘录数据集

处理内表时，也可以使用循环读取摘录数据集的数据：

```
LOOP.
...
[AT FIRST | AT <fgi> [WITH <fgj>] | AT LAST.
...
ENDAT.]
 ...
ENDLOOP.
```

语句 LOOP-ENDLOOP终止创建报表的摘录数据集并在数据集的所有记录上执行循环。 在每个循环过程中， 系统将读取一个摘录记录并将其数据值直接放回到源字段中。 可以连续执行多个循环。

与内表不同，对于摘录数据集不必使用特殊的工作区作为接口。对于每个记录的读取，都可以使用它们原始字段名在循环的语句块中处理读取的数据。

与内表上的 LOOP AT-ENDLOOP循环不同， 在摘录数据集上不能使用嵌套的 LOOP-ENDLOOP循环，否则将产生运行时错误。在循环的语句块中及处理循环后，不允许再使用 EXTRACT 语句，否则将产生运行时错误。

如果只需对数据集的某些记录执行一些语句， 则可使用控制语句 AT 和 ENDAT。系统将针对 AT 不同的选项处理控制语句之间的语句块，如下所示：

60 AT FIRST.

系统将针对数据集的第一条记录执行一次该语句块。

60 AT <fgi> [WITH <fgj>].

如果当前读取的摘录记录的记录类型是用字段组 <fgi> 定义的，系统就处理该语句块。使用 WITH <fgj> 选项时，在摘录数据集中，字段组 <fgi> 当前读取的记录后面必须紧跟字段组<fgj> 的记录。

60 AT LAST.

系统将针对数据集的最后一条记录执行一次该语句块。

13.2.1 排序摘录数据集

和处理内表一样，也可以使用 SORT 语句给摘录数据集排序：

SORT [<order>][AS TEXT]

[BY <f1> [<order>][AS TEXT] ... <fn> [<order>][AS TEXT]].

SORT语句终止创建报表的摘录数据集，同时给数据集记录排序。如果没有 BY 选项，系统将根据在 HEADER字段组中指定的关键字给数据集排序。 要定义不同的排序关键字， 请使用 BY 选项，然后系统将根据指定组件 <f1> ... <fn> 给数据集排序。

SORT语句将摘录过程中未定义的排序关键字中的所有字段按空值对待。系统不管指定的排序顺序如何，总是将空值排在具有定义值的字段的前面。

在处理 SORT 语句以后，不能再执行 EXTRACT 语句。 否则将产生运行时错误。

13.2.2 处理控制集

通过使用 SORT 语句给摘录数据集排序， 可定义一个控制级结构。 摘录数据集的控制级结构对应于 HEADER字段组中的字段的顺序。排序后，可在 LOOP-ENDLOOP循环中使用 AT

语句编写语句块，使系统只在控制中断时才处理它。

AT NEW <f> | AT END OF <f>.

...

ENDAT.

如果字段 <f> 或当前摘录记录中的排序关键字的较高层字段中包含其它值，并且该值与摘录数据集前面的记录（对于 AT NEW）或后续记录（对于 AT END）中的值不同，将产生控制中断，并且系统将处理 AT-ENDAT 中的语句块。字段 <f> 必须是 HEADER字段组的一部分。

13.2.3 计算序号和合计

在使用 LOOP-ENDLOOP读取已排序的摘录数据集时，可以访问两个自动生成的字段 CNT(<f>) 和 SUM(<g>)，这些字段提供了不同值的序号或数字字段的合计。系统将在控制级的最后以及读取数据集的最后一个记录后填充这些字段。如下所示：

61 CNT(<f>)

如果 <f> 是 HEADER字段组的非数字字段，并且系统是根据 <f> 给摘录数据集排序的，则 CNT(<f>) 包含了在控制级中或在整个数据集中分别设定的不同值 <f> 的序号。

61 SUM(<g>)

如果 <g>是摘录数据集的数字字段，SUM(<g>) 将包含控制级或整个数据集中 <g>的值的合计。

十四列表

14.1 自定义列表

14.1.1 定义页眉

要分别定义页眉的格式，必须在紧随事件关键字 TOP-OF-PAGE 的处理块中定义：

TOP-OF-PAGE.

WRITE: ....

TOP-OF-PAGE事件在系统开始处理列表新页时出现。系统在输出新页首行之前处理TOP-OF-PAGE后面的语句。

自定义页眉出现在标准页眉之下。如果要取消标准页眉，使用 REPORT语句的 NO STANDARD PAGE HEADING选项：

REPORT <rep> NO STANDARD PAGE HEADING.

使用该语句时，系统不在报表 <rep> 列表页上显示标准页眉。如果使用 TOP-OF-PAGE 定

义单个页眉，则系统就将其显示出来。

14.1.2 确定列表宽度

要确定输出列表的宽度，使用 REPORT语句的 LINE-SIZE 选项。

REPORT <rep> LINE-SIZE <width>.

该语句根据 <width> 字符确定报表 <rep> 输出列表的宽度。如果将 <width> 设置为 0，则系统使用标准列表的宽度。

14.1.3 确定页长

要确定输出列表的页长，使用 REPORT语句的 LINE-COUNT 选项。

REPORT <rep> LINE-COUNT <length>[(<n>)].

该语句用 <length> 行确定报表 <rep> 输出列表的页长。 如果指定选项号 <n>，则系统为页脚保留 <n>行页长。并不将这些页脚行填写到 END-OF-PAGE 事件中，而是作为空行显示。

14.1.4 定义页脚

要定义页脚，使用 END-OF-PAGE 事件。在处理列表页时，如果系统到达页脚保留行，或者如果 RESERVE语句触发分页则该事件发生。 在紧接着 END-OF-PAGE 事件关键字的处理块中填充页脚的行：

```
END-OF-PAGE.
  WRITE: ....
```

如果为 REPORT 语句的 LINE-COUNT 选项 中的脚 注保留行 ，则系 统只处理 紧接着 END-OF-PAGE的处理块。

14.2 多页列表

14.2.1 编程分页

（1）无条件分页

要在页处理期间触发分页，请使用 NEW-PAGE语句的基本格式：

NEW-PAGE.

该语句结束当前页，在新页上显示所有其他输出。如果输出写入到当前页和 NEW-PAGE 之后的新页，则只开始新页，系统然后将 SY-PAGNO系统字段家 1 ，不能产生空页。

不能触发 END-OF-PAGE 事件，这意味着即使定义了页脚，系统也不输出。

（2）条件分页

要在少于某一页剩余行数的条件下执行分页，使用 RESERVE语句：

RESERVE <n> LINES.

如果当前列表页的最后输出和页脚之间剩余的自由行数少于 <n>，则该语句触发分页。

<n>可以是变量。 在开始新页之前， 系统处理 END-OF-PAGE 事件。只有输出写入到后续页时，RESERVE才生效，不创建空页。

14.2.2 单页的标准页眉

标准页眉包括列表和列标题。 要影响标准页眉这些组件的表现形式， 使用 NEW-PAGE语句的下列选项：

NEW-PAGE [NO-TITLE|WITH-TITLE] [NO-HEADING|WITH-HEADING].

使用 NO-TITLE 或 WITH-TITLE 选项取消或者在以后所有页上显示标准表头， 基本列表的缺省值为 WITH-TITLE，辅助列表为 NO-TITLE。

使用 NO-HEADING或 WITH-HEADING 选项取消或在以后所有页上显示列标题。 基本列表的缺省值为 WITH-HEADING，辅助列表为 NO-HEADING。

即使使用 REPORT 语句的 NO STANDARD PAGE HEADING选项取消标准页眉，也能使用 WITH-TITLE 和 WITH-HEADING 激活单个组件的显示。

14.2.3 单页页长

要分别确定每页的页长，使用 NEW-PAGE语句：

NEW-PAGE LINE-COUNT <length>.

该语句将后续页的页长确定为 <length> 。<length> 可以是变量。 如果将 <length> 设置为 0，则系统使用标准页长。页眉是页的部分，因此也是页长的部分。

14.2.4 所有页页宽

不能更改列表级内单个页宽。 只能更改新列表级的所有页宽。 为此， 使用 NEW-PAGE语句：

NEW-PAGE LINE-SIZE <width>.

从新页开始的所有列表级宽度都为 <width> ， 而不是 REPORT 语句中定义的宽度。

如果将 <width> 设置为 0 , 则系统使用标准列表的宽度。 如果将 <width> 设置为 SY-SCOLS，则可以使新列表级的宽度适配窗口宽度， 即使该窗口小于标准窗口。 SY-SCOLS 系统字段包含当前窗口一行的字符数。

14.2.5 在程序之内滚动

在程序之内，可以垂直和水平滚动列表。使用 SCROLL 关键字。 SCROLL语句只对完成的列表生效。 如果在列表的第一条输出语句之前使用该语句， 则不影响该列表。 如果在列表的第一条输出语句之后使用 SCROLL，则影响整个列表，包括以后的所有输出语句。

在每条 SCROLL 语句之后， 可以查询 SY-SUBRC 以查看系统是否成功。 如果系统成功滚动则 SY-SUBRC 为 0 ；如果滚动不可能则为 4 ，因为其超过列表边界。

（1）通过窗口滚动

要以当前窗口大小垂直滚动列表并且与页长无关，使用如下语句：

SCROLL LIST FORWARD|BACKWARD [INDEX <idx>].

不带 INDEX 选项时， 该语句以当前窗口大小向前或向后滚动当前列表。 使用 INDEX 选项时，系统以列表级 <idx> 滚动列表。

（2）按页滚动

要按页滚动列表，即按页长垂直滚动列表， SCROLL语句提供下列选项：

1）滚动到某页

要滚动到某页，使用 SCROLL 语句的 TO 选项：

SCROLL LIST TO FIRST PAGE | LAST PAGE | PAGE <pag>

[INDEX <idx>] [LINE <lin>].

不用 INDEX 选项，语句将当前列表滚动到首页、尾页或 <pag>页。使用 INDEX 选项，则系统滚动列表级 <idx> 的列表。 使用 LINE 选项时， 系统显示从实际列表的 <lin> 行开始滚动的页，不包括页眉行。

2）按页数滚动

要按页数滚动列表，使用 SCROLL 语句的下列选项：

SCROLL LIST FORWARD | BACKWARD <n> PAGES [INDEX <idx>].

不用 INDEX 选项时，语句向前或向后滚动 <n>页。

3）滚动到列表的页边缘

要水平滚到列表的左或右页边缘，使用 SCROLL 语句的下列选项：

SCROLL LIST LEFT | RIGHT [INDEX <idx>].

不用 INDEX 选项时， 语句滚动到当前列表的左或右页边缘。 使用 INDEX 选项时， 系统滚动列表级 <idx> 的列表。

（3）按列滚动

要按列水平滚动列表， SCROLL 语句提供几个选项。 在此情况下， 一列意味着列表行一个字符。

1）滚动到某列

要滚动到某列，使用 SCROLL 语句的 TO COLUMN选项：

SCROLL LIST TO COLUMN <col> [INDEX <idx>].

不用 INDEX 选项时，系统显示从列 <col> 开始的当前列。使用 INDEX 选项时，系统滚动列表级 <idx> 的列表。

2）按列数滚动

要按某一列数滚动列表，使用 SCROLL 语句的下列选项：

SCROLL LIST LEFT | RIGHT BY <n> PLACES [INDEX <idx>].

不用 INDEX 选项时，系统按 <n>列将当前列表滚动到左边或右边。

14.3 列表页面设置

14.3.1 定位输出

可以在当前页的任意位置定位 WRITE 和 ULINE 语句的输出。紧接着位置规格的 WRITE、SKIP 或 ULINE 语句可以覆盖现有输出。

65 SY-COLNO （用于当前列）

65 SY-LINNO （用于当前行）

可以使用系统字段在页上引导。ABAP/4 提供一些关键字以更改绝对和相对输出位置。

（1）绝对定位

指定绝对位置之后，将后续输出写入在固定行和列开始的屏幕。

1）水平定位

要指定水平输出位置，ABAP/4 提供两种方法：

WRITE 和 ULINE 语句的 AT 选项以及 POSITION 语句。POSITION 语句的语法为：

POSITION <col>.

该语句将水平输出位置和 SY-COLNO 系统字段设置为 <col> 。如果 <col> 超出页面之外，则忽略后续输出语句。系统将 POSITION 语句或使用 AT 格式化的 WRITE 语句的输出写入指定位置，不管是否有足够空间。超出该行的输出部分被截断。其他 WRITE 输出在下一行开始。

2）垂直定位

如下指定垂直输出位置：

SKIP TO LINE <n>.

该语句将垂直输出位置和 SY-LINNO 系统字段设置为 <lin> 。如果 <lin> 超出 1 和页长范围，则系统忽略该语句。

3）在页眉之下定位

要将输出定位到整个页眉之后的第一行，使用 BACK 语句：

BACK.

如果该语句不是紧跟 RESERVE语句，则后续输出出现在页眉之下。系统将 SY-COLNO 设置为 1 并根据页眉的长度设置 SY-LINNO。如果在 TOP-OF-PAGE 事件中指定 BACK，则系统

不将输出位置设置到整个页眉之下，而只在标准页眉之下。写入的任何输出现在都覆盖TOP-OF-PAGE中指定的自定义页眉。

（2）相对定位

相对定位参考以前写入列表的输出。 某些相对定位自动发生。 使用无定位的 WRITE 时，在上一输出之后空一列出现输出。 如果在当前行没有足够空间， 则换行。不使用定位的 ULINE 和 SKIP 语句产生换行。

1）产生换行

要产生换行，可在 WRITE、 ULINE 或 NEW-LINE 语句的 AT 选项中使用反斜杠。

NEW-LINE.

该语句将输出定位在新行中，将 SY-COLNO设置为 1 并且将 SY-LINNO 加 1。如果自最后换行之后将输出写入屏幕， 则系统只执行该语句。 NEW-LINE 不创建空行。 在 NEW-PAGE 语句和事件开始时发生自动换行。

2）将输出定位到其他输出之下

可以将 WRITE 输出定位到上一 WRITE 输出的列中。使用 WRITE 语句的格式化选项 UNDER。

WRITE <f> UNDER <g>.

系统从开始输出字段 <g>的同一列中开始输出 <f>。该语句不限于当前页，即 <g>不必出现在相同页上。

3）将输出定位在行块的首行

要将下一输出行定位到通过 RESERVE语句定义的行块的首行，按如下格式使用 BACK 语句：

RESERVE.

.....

BACK.

如果 BACK 紧随 RESERVE， 则后续输出出现在 RESERVE之后的第一行中。

14.3.2 格式输出

（1） FORMAT语句

要在程序中静态设置格式化选项，按如下格式使用 FORMAT语句：

FORMAT <option1> [ON|OFF] <option2> [ON|OFF] ....

FORMAT语句中设置的格式化选项 <optioni> 适用于所有后续输出，直到使用 OFF 选项关闭。

要在运行时动态设置格式化选项，按如下格式使用 FORMAT语句：

FORMAT <option1> = <var1><option2> = <var2> ....

1）列表中的颜色

FORMAT语句的选项 COLOR、INTENSIFIED 和 INVERSE 影响输出列表的颜色。要在程序中设置颜色，使用：

FORMAT COLOR <n> [ON] INTENSIFIED [ON|OFF] INVERSE [ON|OFF].

要在运行时设置颜色，使用：

FORMAT COLOR = <c> INTENSIFIED = <int> INVERSE = <inv>.

这些格式化选项不适用于由 ULINE 创建的水平行。其功能如下所示：

COLOR设置行背景的颜色，如果设置 INVERSE ON，则系统将更改前景颜色而不是背景颜色。下表汇总各种不同的选项：

	<n>	<c>	颜色	用于
OFF	或 COL_BACKGROUND	0	取决于 GUI	背景
1	或 COL_HEADING	1	灰蓝	标题
2	或 COL_NORMAL	2	淡灰	列表正文
3	或 COL_TOTAL	3	黄	总计
4	或 COL_KEY	4	蓝绿	关键字列
5	或 COL_POSITIVE	5	绿	正门限值
6	或 COL_NEGATIVE	6	红	负门限值
7	或 COL_GROUP	7	紫	组级别

缺省值为 COLOR OFF。INTENSIFIED 确定行背景的颜色调色板。

2）使字段可接受输入

要在程序中将输出字段设置为“准备输入”，按如下格式使用 FORMAT语句：

FORMAT INPUT [ON|OFF].

要在运行时将输出字段设置为“准备输入”，使用：

FORMAT INPUT = <i>.

3）使字段输入为热点

要将某一区域输出为热点，使用 FORMAT语句的下列选项：

FORMAT HOTSPOT [ON|OFF].

要在运行时将字段设置为热点，使用：

FORMAT HOTSPOT = <h>.

14.3.3 特殊格式输出

（1）国家或用户特有格式输出

使用下列语句可以在程序中更改这些设置：

SET COUNTRY <c>.

（2）货币格式输出

要根据货币格式化数字字段的输出，使用 WRITE 语句的 CURRENCY 选项：

WRITE <f> CURRENCY <c>.

（3）单位特有格式输出

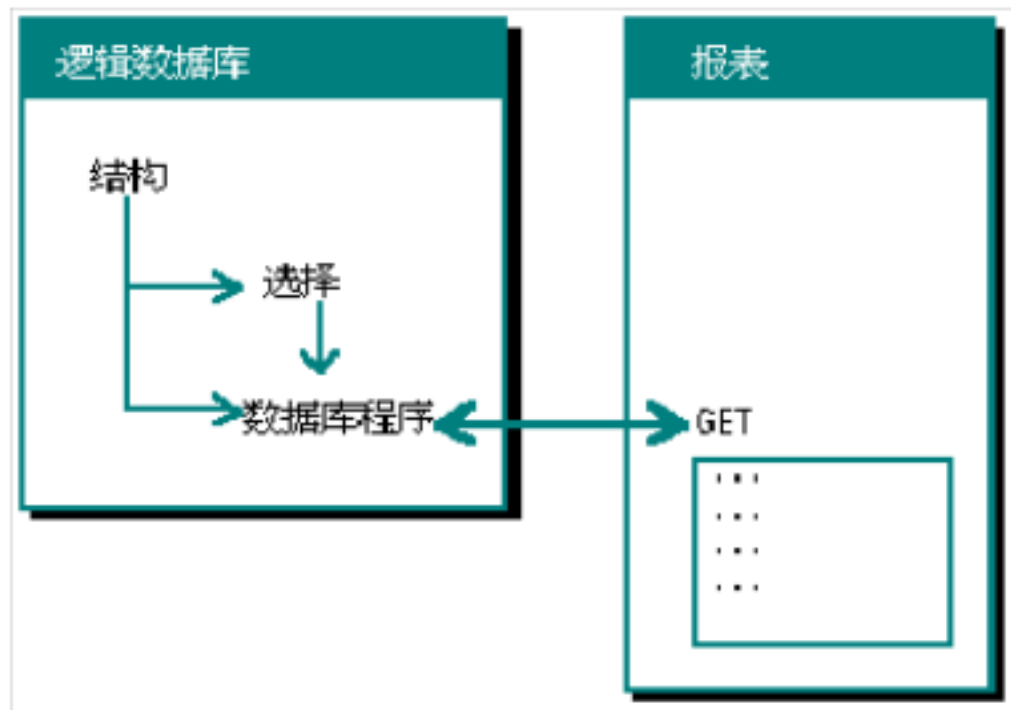
可以根据某一单位格式化类型 P 字段，使用 WRITE 语句的 UNIT 选项：

WRITE <f> UNIT <u>.

该语句根据单位 <u>设置小数位。

十五逻辑数据库

ABAP/4 报表使用逻辑数据库读取和处理数据。报表可用的数据顺序取决于相关逻辑数据库的层次结构。 逻辑数据库也提供用户对话框 （即选择屏幕） 的界面并且将用户输入检查和错误对话组合起来，可以在报表中修改并扩展该界面。



逻辑数据库包括至少下列三个组件：

（1）结构

结构是逻辑数据库的基本组件。它决定其它组件的结构以及运行时逻辑数据库的行为。

（2）选择

该组件决定每个报表的用户界面， 其格式通常由结构决定。 可以调整和扩展选择以适应

需要。

（3）数据库程序

数据库程序是选择数据并将其传递到报表的子程序集合。 数据库程序的格式由结构和选择共同决定，可以调整和扩展数据库程序以适应需要。

15.1 结构

通常，逻辑数据库反映 SAP 系统中层次表格的外来关键字相关性。逻辑数据库有定义如下的层次结构：

（1）最高层只有一个节点，称为根节点。

（2）每个节点可以有一个或几个分支。

（3）每个节点从其它节点派生。

15.2 选择

在逻辑数据库中， 可以使用 SELECT-OPTIONS和 PARAMETERS语句在选择屏幕上定义输入字段。通过称为选择包含程序的特定包含程序的帮助可达此目的。 在每个 ABAP/4 报表中，可以通过定义报表特定选择扩展逻辑数据库选择。 所有报表特定选择都显示在特定数据库选择之后。

生成报表的选择屏幕时， 系统只考虑数据库的特定选择标准和参数， 其相应表格由报表中的 TABLES 语句声明。

15.2 数据库程序

逻辑数据库 <dba>的数据库程序的名称符合命名规则 SAPDB<dba>。重要的是， 它包括系统在运行 ABAP/4 报表时调用的子程序集合。 报表中带事件关键字的子程序之间的相互作用，在逻辑数据库和 ABAP/4 报表中进行了描述。

逻辑数据库程序通常包含定义子程序中描述的下列子程序， 并且都使用 FORM 语句定义。

（1） FORM INIT

在显示选择屏幕之前调用一次。

（2） FORM PBO

每次刷新选择屏幕之前调用。

（3） FORM PAI

用户每次在选择屏幕上按 ENTER 时调用。

系统将参数 FNAME 和 MARK 传递给子程序，这些参数自动定义和填充。

（1） FNAME包含选择屏幕上选择标准或参数的名称。

（2） MARK说明用户所做的选择：

MARK = SPACE意味着用户已经输入简单的单值或范围选择。

MARK = '*'　意味着用户已经在“ 多重选择 ”屏幕上制作条目。

（3） FORM PUT_<table>

以逻辑数据库结构决定的顺序调用子程序。　使用 SELECT 语句读取节点 <table> 的数据, 并且 PUT 语句将程序流定向到报表中合适的　GET 语句中。

PUT 语句是该子程序的主语句：

PUT < table >.

只能在逻辑数据库子程序中使用　PUT <table> 语句, 该逻辑数据库包含以　PUT_< table　> 开头的名称的节点　<table> 。

15.3　逻辑数据库的授权检查

通常，可以在下列数据库程序的子程序或者报表的处理块中包括授权检查：

数据库程序中的子程序：

- PAI
- AUTHORITY_CHECK_<table>

报表中的事件关键字：

- AT SELECTION-SCREEN
- AT SELECTION-SCREEN ON <fname>
- AT SELECTION-SCREEN ON END OF <fname>
- GET <table>

在任何情况下, 数据库访问和应用逻辑的分离允许在逻辑数据库程序中集中地编码所有授权，这使维护大的编码系统更加容易。

15.4　逻辑数据库的性能

因为在所有有关的　ABAP/4 报表中更改逻辑数据库将立即生效,　所以通过集中优化可以提高程序库中的不同对象的响应时间。

通过允许用户精确指定系统从数据库中读取哪个表格条目可以获得最大的性能提高。　为此，可以在数据库程序中使用下列技术：

（1）选择标准和参数，可能带默认值和值列表。

（2）动态选择。

（3）匹配码选择。

（4）查看从数据库读取的条目或者将其存储在内表中。

另外, 应该在早期进行授权检查,　即尽可能在选择屏幕处理期间而不是等到数据选择处

理期间。因为它们依赖于读取的数据，因此没有优化的步骤规则。

十六事务

16.1 对话编程

16.1.1 事务

事务是实施与用户对话的程序。 在典型的对话中， 用户可以在系统显示的屏幕上输入或请求信息。 作为对用户输入或请求的响应， 程序执行适当的动作：转到下一个屏幕， 显示输出或更改数据库。

16.1.2 对话程序的结构

对话程序包含下面的基本组件：

(1) 屏幕（动态程序）

SAP 系统中的每个对话都是由动态程序控制的。动态程序包含一个屏幕和它的流逻辑 ，并且精确控制一个对话步骤。流逻辑决定在显示屏幕前（输出前的 PBO 处理）和接收到用户在屏幕上制作的条目后（输入后的 PAI 处理）进行的处理。

在屏幕绘制器中固定的屏幕格式决定输入 / 输出字段、文本字段和诸如单选按钮以及复选框之类的图形元素的位置。另外，菜单绘制器允许在一个或多个 GUI 状态下存储菜单、图表、 按钮及功能键。 动态程序和 GUI 状态都指向控制动态程序和 GUI 状态的运行顺序的 ABAP/4 程序。

（2） ABAP/4 模块池

每个动态程序都精确指向一个 ABAP/4 对话程序。 这样的对话程序也称为模块池， 原因是它包含交互模块。动态程序的流逻辑包含从相应模块池中调用模块。在 PBO 事件中调用的交互模块依照上下文进行屏幕模板的准备， 例如通过设置字段内容或通过禁止显示不需要的字段等方式准备屏幕模板。在 PAI 事件中调用的交互模块用于检查用户的输入并触发合适的对话步骤，例如更新任务。

从一个事务中调用的所有动态程序都指向一个公用模块池， 模块池的动态程序是编号的。默认情况下， 系统为每个动态程序存储下一个要显示的动态程序。 该动态程序序列或链可以是线性的也可以是环型的。 在动态程序链中甚至可以调用另一个动态程序链， 并且在处理它之后返回原始。

16.1.3 传输字段数据

与报表编程相反，不能用 WRITE 语句将字段数据写入屏幕。系统通过屏幕字段名和 ABAP/4 变量名的比较来代替数据传输。 如果两个名称相同， 它将屏幕字段值传输给 ABAP/4

程序字段，反之亦然。在显示屏幕之前和之后立即发生上述过程。

16.1.4 字段属性

在屏幕制作器中定义所有动态程序屏幕字段的属性。如果屏幕中的字段名对应于某个 ABAP/4 词典字段的名称，则系统自动建立这两个字段之间的参考，因此自动从 ABAP/4 词典中复制动态程序中的大量字段属性。 字段属性和数据元素以及分配的词典字段形成了动态程序在对话中执行的标准函数的基础。

16.1.5 错误对话

动态程序处理器的另一个任务是管理错误对话。 使用 ABAP/4 词典的检查表格自动检查输入数据或者通过 ABAP/4 程序本身检查。 动态程序处理器包括接收屏幕中的错误消息并将屏幕返回用户。 消息可以是上下文敏感的， 也就是说， 系统用当前字段内容替换消息文本中的占位符。另外，只有内容与错误有关并且可以纠正错误的字段才可以接收输入。

16.1.6 数据一致性

要在复杂的应用程序中保持数据一致性， ABAP/4 提供优化数据库更新的技术，对它的操作独立于下面的数据库并符合对话编程的特殊请求。

16.2 动态程序

每个屏幕都包含用于显示或请求信息的字段。 字段可以是文本串、 输入或输出字段、 单选按钮、复选框或按钮。

SAP 动态程序包含几个组件：

（1）流逻辑：为屏幕调用 ABAP/4 模块。

（2）屏幕格式：屏幕上文本、字段、按钮等的位置。

（3）屏幕属性：屏幕号、后续屏幕号及其它。

（4）字段属性：屏幕上单独字段的属性定义。

16.3ABAP模块池

在对象浏览器中，模块池代码属于下列类别之一：

（1）全局字段：模块池中所有模块都可使用的数据声明；

（2）PBO 模块：显示屏幕前调用的模块；

（3）PAI 模块：响应用户输入而调用的模块；

（4）子程序：可以在模块池中任何位置调用的子程序。

默认情况下， 系统将模块池分成一个或多个包含程序。 一个包含程序可以包含相同类型的多个模块（仅 PBO 模块或 PAI 模块），然后主程序包含一系列将模块链接到模块池的 INCLUDE 语句。