# Workbench Transactions

**SE38 –** ABAP Editor

**SE80 –** Runtime repository

**Symbol used in programs**

Period operator ‘**.**’ - Statement conclusion

Colon ‘**:**’ – statement continuous

Comma ‘,’ – Separator

Asterisk ‘**\***’ – Comment line – should start from first column

Double Quote ‘**”**’ – Inline comment line may start from any column. Generally used for 40th line comment line

Single Quote ‘**’**’ - Literal enclosure

# WRITE statement

This statement is used to print static text or content of variable on screen. It can also be used to copy the content of one variable to another variable.

|  |  |
| --- | --- |
|  |  |
| Write the Same Program using Chain Operator |  |

## Write statement additions

1. New line **'/'** character initiates output on next line.
2. Chain Operator **':'** and Separator **','** Chain operator is used to combine several similar statements which have same prefix. In the above example WRITE statement. So we can combine above two statements.
3. Output location and sizewe can specify location from where output should start printing and length of the output.

|  |  |
| --- | --- |
| Output printed on different lines. |  |
|  |  |

# DATA TYPES and DATA OBJECTS

Data Types defines technical attributes of data. Whereas Data Objects which are declared with help of Data Types

Based on its complexity we can classify into following types

* Elementary Data Types
* Structured Types
* Table Types

## Elementary and Variables

Elementary data types are used to declare variables which hold single value. There different types of Elementary Types

* Built In Data Type
* Data Elements
* Table Fields
* User Defined Types

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Built in Data type: SAP has given some data types called as built in data types.     |  |  |  |  | | --- | --- | --- | --- | | **Type** | **Length** | **Standard length** | **Description** | | B | 1 Byte |  | 1 byte integer (internal) | | C | 1 to 65,535 characters | 1 character | Text field | | Cursor | as i | As i | Database cursor | | D | 8 characters |  | Date field | | F | 8 bytes |  | Floating point number | | I | 4 bytes |  | 4 byte integer | | N | 1 to 65,535 characters | 1 character | Numeric text | | P | 1 to 16 bytes | 8 bytes | Packed number | | String | Variable |  | Text string | | S | 2 bytes |  | 2 byte integer (internal) | | T | 6 characters |  | Time field | | X | 1 to 65,535 bytes | 1 byte | Byte field | | xstring | Variable |  | Byte string | | |
|  |  |
| Declare Variables using Data Elements | |
|  |  |
| Declare Variables using Table Fields | |
|  |  |
| Declare Variables using User Define Types  We can declare our own data types using T**YPES** statement. Structures, Tables, Data elements defined in Data Dictionary will be used as data types to declare variables. | |
|  |  |

## Structured Types and Work Areas:

Structured data types are composed of several simple data types or other structure data types are used to declare work areas which holds group of values

Following are the Structure Types used to declare work areas

* Tables
* Dictionary Structures
* User Defined Structures

|  |  |
| --- | --- |
| Declare Work area based on Database Table | |
|  |  |
| Declare Work area based on Dictionary Structure | |
|  |  |
| **Structured data types and data objects**  **Syntax**  TYPES : BEGIN OF <struct\_type>,  <field\_1> type <type\_1>,  <field\_2> type <type\_2>,    <field\_n> type <type\_n>,  END OF <struct\_type>  Above is the user defined structured data type, which used to declare multiple structured data objects | |
|  |  |

## Table types and Internal Tables

# Internal tables and Operations

Internal tables are multi line structured types/objects. We can store several lines of similar structured data in an internal table. While defining internal table, we can specify some characteristics, so that it behaves in that way. we can specify key (Unique/Non Unique) for internal table. We can declare internal table by referring ABAP dictionary structure, Local structure or type pool structures.

## Types of internal tables:

* Standard - Most frequently use
* Sorted - data will be stored in sorted order so no need to sort again
* Hashed -

## Operations on Internal tables

### Filling

* Append/ Append lines of
* Insert / Insert lines of
* Collect

### Retrieving

* Loop - loop through entire internal table record by record
* Read - search internal table

### Delete

* Clear/Refresh - delete all records without specifying any condition
* Delete - delete records based on condition

### Sort

Sort internal table

### Describe –

Count number of lines in internal table

# Operators

|  |  |
| --- | --- |
| **Operator** | **Description** |
| =, EQ | Equal: True, if the content of operand1 matches the content of operand2. |
| <>, NE | Not Equal: True, if the content of operand1 does not match the content of operand2. |
| <, LT | Lower Than: True, if the content of operand1 is smaller than the content of operand2. |
| >, GT | Greater Than: True, if the content of operand1 is greater than the content of operand2. |
| <=, LE | Lower Equal: True, if the content of operand1 is lower than or equal to the content of operand2. |
| >=, GE | Greater Equal: True, if the content of operand1 is greater than or equal to the content of operand2 |

**Between:**

A logical expression with the language element BETWEEN checks where an interval belongs

Syntax ... operand [NOT] BETWEEN operand1 AND operand2 ...

**IS**

It is Logical expressions with the language element IS check the state of an operand.

**IN**

In a logical expression with language element IN, the conditions of a [selection table](SAPEVENT:%22ABENSELECTION_TABLE_GLOSRY%22) are checked.

**AND, OR, NOT** –

The Boolean operators [AND](SAPEVENT:%22ABENLOGEXP_AND%22) & [OR](SAPEVENT:%22ABENLOGEXP_OR%22) link, in each case, multiple logical expression, while the operator [NOT](SAPEVENT:%22ABENLOGEXP_NOT%22) negates a logical expression. Logical expressions can be explicitly bracketed with [( )](SAPEVENT:%22ABENLOGEXP_BRACKET%22).

Note: example will be available in further sections

# Control structures

## Controls flow of Program execution. We have three kinds of Control Structures

* Branching
* Looping
* Breaking

## Branching blocks:

These statements execute block of statements based on result of condition.

|  |  |
| --- | --- |
| IF…ELSEIF….ENDIF. **IF log\_exp1.**  **[statement\_block1]**  **[ELSEIF log\_exp2.**  **[statement\_block2]]**  **...**  **[ELSE.**  **[statement\_blockn]]**  **ENDIF.** | |
|  |  |
|  |  |
| Example#1 |  |
| Example#2 |  |
| CASE .. WHEN … ENDCASE. **CASE operand.**  **[WHEN operand1 [OR operand2 [OR operand3 [...]]].**  **[statement\_block1]]**  **...**  **[WHEN OTHERS.**  **[statement\_blockn]]**  **ENDCASE.** | |
| Report ZSAMPLE\_CASE\_ENDCASE.  Data l\_input type I value 1.  Case l\_input.  When 1.  Write ‘Given values is one’.  When 2.  Write ‘Givenvalue is two’.  When 3.  Write ‘Given value is Three’.  When others.  Write ‘Given values is other than1 2 3’.  Endcase. |  |

## Looping blocks:

These statements executes block of statements repeatedly based on condition.

### DO … ENDDO

**DO [n TIMES] [**[**varying**](SAPEVENT:%22ABAPDO_VARYING%22)**].   
[statement\_block]   
ENDDO.**

|  |  |
| --- | --- |
| DO 5 TIMES.  Write / sy-index.  ENDDO. |  |

### WHILE…ENDWHILE.

**WHILE log\_exp [**[**vary**](SAPEVENT:%22ABAPWHILE_VARY%22)**]**   
**[statement\_block]**   
**ENDWHILE.**

|  |  |
| --- | --- |
| While sy-index LE 5.  Write sy-index.  Endwhile |  |

|  |  |
| --- | --- |
| data it\_mara type table of mara.  Referring local structure   * Declare local structure type   types :  begin of ty\_mara,  matnr type matnr,  mtart type mtart,  matkl type matkl,  end of ty\_mara.   * Declare internal table   Data: it\_mara type table of ty\_mara, “ internal table  Wa\_mara type ty\_mara. “ work area  Data :l\_lines type i.   * Append data to internal table   Wa\_mara-matnr = ‘M1’.  Wa\_mara-mtart = ‘HALB’.  Wa\_mara-matkl = ‘AB’.  Append wa\_mara to it\_mara.  Clear wa\_mara.  Wa\_mara-matnr = ‘M3’.  Wa\_mara-mtart = ‘HALB’.  Wa\_mara-matkl = ‘CD’.  Append wa\_mara to it\_mara.  Clear wa\_mara.  Wa\_mara-matnr = ‘M2’.  Wa\_mara-mtart = ‘HALB’.  Wa\_mara-matkl = ‘BC’.  Append wa\_mara to it\_mara.  Clear wa\_mara.   * Sorting internal table   Sort it\_mara by matnr.   * Looping internal table   Describe table it\_mara lines l\_lines.  Write : ‘Number of records before clear ’, l\_lines.  Write : /2 ‘Material’, 15 ‘Type’, 24 ‘Group’.  Loop at it\_mara into wa\_mara.  Write: /2 wa\_mara-matnr, 15 wa\_mara-mtart, 24 wa\_mara-matkl.  Endloop.   * Reading a single record from internal table   Read table it\_mara into wa\_mara with key matnr = ‘M1’.  Write : / ‘Material :’, wa\_mara-matnr,  / ‘Type :’, wa\_mara-mtart,  / ‘Group :’, wa\_mara-matkl.  Clear it\_mara.  Describe table it\_mara lines l\_lines.  Write : ‘Number of records after clear ’, l\_lines. |  |

## Internal table control statements and Operations inside internal table

1. AT NEW <fld> .. ENDAT.

Statements written inside this block will be executed for every new value of <fld>. If value of <fld> is repeated then it won’t be executed

1. AT END OF <fld>.. ENDAT.

Statements written inside this block will be executed for end of existing value of <fld>.

1. ON VARRYING <fld\_list>.. ENDON. “obsolete

Statements written inside this block will be executed on every change of value of fields.

Do: It is recommended to sort internal table before looping.

Above statements are used inside loop at <itab>….endloop.

|  |  |
| --- | --- |
| * Declarations   Types:  Begin of ty\_emp,  State(10) type c,  City(25) type c,  Num type i,  Name(20) typec,  End of ty\_3mp.  Data:  it\_emp type table of ty\_emp, “ internal table  Wa\_emp type ty\_emp. “ work area   * Fill data.   Wa\_emp-state = ‘AP’.  Wa\_emp-city = ‘HYD’.  Wa\_emp-num = 1.  Wa\_emp-name = ‘ABCD’.  Append wa\_emp to it\_emp.  Wa\_emp-state = ‘AP’.  Wa\_emp-city = ‘VWD’.  Wa\_emp-num = 2.  Wa\_emp-name = ‘ABCD’.  Append wa\_emp to it\_emp.  Wa\_emp-state = ‘AP’.  Wa\_emp-city = ‘HYD’.  Wa\_emp-num = 3.  Wa\_emp-name = ‘ABCD’.  Append wa\_emp to it\_emp.  Wa\_emp-state = ‘AP’.  Wa\_emp-city = ‘VIZ’.  Wa\_emp-num = 4.  Wa\_emp-name = ‘ABCD’.  Append wa\_emp to it\_emp.  Wa\_emp-state = ‘TN’.  Wa\_emp-city = ‘KOB’.  Wa\_emp-num = 1.  Wa\_emp-name = ‘ABC’.  Append wa\_emp to it\_emp.  Wa\_emp-state = ‘TN’.  Wa\_emp-city = ‘CHN’.  Wa\_emp-num = 2.  Wa\_emp-name = ‘ABD’.  Append wa\_emp to it\_emp.  Wa\_emp-state = ‘UP’.  Wa\_emp-city = ‘LUK’.  Wa\_emp-num = 3.  Wa\_emp-name = ‘BCD’.  Append wa\_emp to it\_emp.  Wa\_emp-state = ‘UP’.  Wa\_emp-city = ‘LUK’.  Wa\_emp-num = 8.  Wa\_emp-name = ‘ACD’.  Append wa\_emp to it\_emp.  Here there are 8 records in internal table  Sort it\_emp by state.  **LOOP AT** it\_emp **INTO** wa\_emp. “ Internal table loops 8 times as there are 8 records  **AT NEW** state.  **WRITE :** / ‘State :--🡪’, wa\_emp-state.  “ this statement block is executed for every new value of state. In this example it will be “executed 3 times only even there are 8 records in internal table, because there are 3 distinct values (AP,TN and UP) in state field in internal table.  **ENDAT**.  “ this block will be executed 8 times.  **WRITE** : / wa\_emp-num, wa\_emp-name.  **ENDLOOP.** |  |

Output:

State :-🡪 AP

1 abcd

2 abcd

3 abcd

4 abcd

State:-🡪TN

1 abc

2 abd

State -🡪 UP

3 BCD

8 ABD

Important note:

Suppose there are 5 fields in an internal table. Let say fld1,fld2,fld3,fld4,fld5.

…..still continue ….

## Copying data

We can copy data from one variable/work area/internal table to another variable/work area/internal table using following operators

1. = “elementary/work area/internal table
2. Write <src> to <dest> “ elementary
3. Move <src> to <dest> “ elementary/work area
4. Move corresponding <src> to <dest> “ work area

Assign <src> to <dest> . “ working with field symbols

# Open SQL

Database: Database is a permanent storage area.

Relational database: data will be stored in the form of tables (Rows and columns).

SQL – Structured Query Language is used to work with data base. It has some set of statements. These statements are used to

**Working with table structure**

* Creating tables means defining structure of table; consist of column names, type and its length, imposing conditions on data to be inserted.
* Altering tables means adding/deleting columns, changing conditions
* Deleting table means delete of table

**Working with table data**

* Inserting/modifying/updating rows in a table
* deleting rows in a table
* Reading rows from a table

**Working with table locks**

* Read lock
* Write lock

**Open SQL:** SAP proprietary SQL statements. These are subset of standard SQL statements. Open SQL statements are used to working with table data (Insert/Modify/Update/Read/Delete). Creating/ Modifying/Deleting tables are being done in transaction **SE11- data dictionary.** SAP is taking care of locking mechanism. But sometimes we need to take care of locking table in our own applications.

## Open SQL statements

Reading data from database table

* Single record
* Multiple records
* Aggregate results

Inserting data to database table

* Inserting single record
* Inserting multiple records
* Inserting data to multiple tables via maintenance views.

Modifying data in the database table

* Modifying single record
* Modifying multiple records
* Modifying data in multiple tables

Deleting data

* Deleting single record
* Deleting multiple records

|  |  |
| --- | --- |
| SELECT Statement **Select Single record and single/multiple fields**  **Destination as individual variables** | |
| Data: l\_matnr type matnr, “ data elements defined in data dictionary  L\_mtart type mtart.  Select single matnr mtart  From mara  Into (l\_matnr,l\_mtart).  Write : l\_matnr,l\_mtart. |  |
| Above statement reads first record from the table. If you want get specific record use **WHERE** clause. Using **WHERE** we can specify some condition.  Example:  Select single matnr mtart  From mara  Into (l\_matnr,l\_mtart)  Where matnr = ‘M1’. |  |
| Note: we should take care of order of retrieval and destination. Otherwise it will get short dumps. Otherwise use **CORRESPONDING** addition to the **INTO** clause.  Select matnr mtart  From mara  Into corresponding (l\_mtart,l\_matnr).  From the above select statement; in **INTO** I have specified order (mtart,matnr) in different to selection order(matnr,mtart).  **Destination as work area**  Data: wa\_mara type mara.  Select single \*  From mara  Into wa\_mara  Where matnr = ‘M1’.  Most of the times, **SELECT SINGLE** is used for existence checking of record in table. It is recommended to use primary key in where clause. |  |
| **Select multiple records single fields/multiple fields** | |
| This statement is used to fetch multiple records from single/multiple tables.  **Example:**  Selecting all fields  Data: it\_mara type table of mara.  Select \*  From mara  Into table it\_mara  Where mtart = ‘HALB’.  Above statement get all half finished goods from MARA table.  Selecting specific set of fields  Data:  Begin of wa\_mara,  Matnr type matnr,  Mtart type mtart,  Matkl type matkl,  End of wa\_mara,  It\_mara like table of wa\_mara.  We can write select statement in the following ways |  |
| |  |  |  | | --- | --- | --- | | **SELECT**  Matnr  Mtart  Matkl  **FROM** mara  **INTO TABLE** it\_mara  **WHERE** mtart = ‘HALB’.  Note: this approach is recommended. Specify columns selecting order same as work area fields order. | **SELECT** \*  **FROM** mara  **INTO CORRESPONDING FIELDS OF TABLE** it\_mara  **WHERE** mtart = ‘HALB’.  Note: This approach has performance issues | **SELECT**  Mtart  Matkl  Matnr  **FROM** mara  **INTO CORRESPONDING FIELDS OF TABLE** it\_mara  **WHERE** mtart = ‘HALB’.  Note: This is just for example but not used | | |
| **FOR ALL ENTRIES IN <itab> and WHERE clause**  Note: now we can process above internal table based on requirement |  |
| 1. **Aggregate results**   Aggregate functions are worked on group of records. |  |
| 1. **Selecting specific number of records** 2. Use **UP TO <n> ROWS** clause for fetching <n> rows from the table |  |
| 1. **Get data in ascending/descending order** 2. Use **ORDER BY** clause to get data in specific order |  |
| Example:  Data:  Begin of wa\_mara,  Matnr type matnr,  Mtart type mtart,  Matkl type matkl,  End of wa\_mara,  It\_mara like table of wa\_mara.  **SELECT**  Matnr  Mtart  Matkl  **FROM** mara  **UP TO** 10 **ROWS**  **INTO TABLE** it\_mara  **WHERE** mtart = ‘HALB’  **ORDER BY** matnr **ASCENDING**. |  |
| Note: Above select query first filters records which are HALB finished goods then it sorts in ascending order and then returns first 10 records |  |
| INSERT  MODIFY/UPDATE  DELETE |  |

# Modularization

It is application/program development approach, facilitates us easy development and maintenance of the program/application.

Moto: **Define once and use many times.**

It can be achieved by following ways

1. Macros – replacement mostly used in HR module
2. Includes - replacement
3. Subroutines - reusability
4. Function Modules – reusability

Replacement – It means, Part of program will be kept in some other location and will be replaced during execution.

Reusability - Part of the code will be kept in some other location. If you call it, execution will be done in there only. Only result will be sent.

|  |  |
| --- | --- |
| Report ZSAMPLE\_INCLUDE\_01.  Data:  l\_num1 type I,  l\_num2 type I,  l\_result type i.  l\_result = l\_num1 + l\_num2.  Write : / ‘Sum of ’,l\_num1 , ‘and ’,l\_num2, ‘ is’,l\_result.  Rewrite above program using **INCLUDE** | Program ZSAMPLE\_INCLUDES\_01.  Include zsample\_include\_top.  L\_result = l\_num1 + l\_num2.  Write : / ‘Sum of ’,l\_num1 , ‘and ’,l\_num2, ‘ is’,l\_result.  \*\*\* **include program**  Include zsample\_include\_top.  Data:  l\_num1 type I,  l\_num2 type I,  l\_result type i. |

## Subroutine

Subroutines are processing blocks contains some business logic, which can be executed when we call it. We can pass values (elementary variables, work areas, internal tables) to the subroutines.

Syntax:

### Calling subroutine:

Perform <sub\_routine> tables <send\_and\_received\_tables>

Using <send\_variables>

Changing <send\_and\_received\_variables>.

### Defining subroutine:

Form <sub\_routine> tables <send\_and\_received\_tables>

Using <received\_variables>

Changing <send\_and\_received\_variables>.

“ logic

Endform.

### Types of parameters

Actual Parameters – values which provided from main program while calling subroutine

Formal Parameters – values which available at subroutine at definition of subroutine

**Passing/Getting values to/from subroutine**

Passing via Using/changing/tables clauses

Getting via changing/tables clauses

**Mechanisms**

Pass by value – actual parameter retains its own value.

Pass by reference – actual parameters value can be changed

Pass by value/reference – actual parameter only changed when complete execution of subroutine

Note:

1. Variable, which declared in main program, will be automatically available in all subroutines, which are defined in the program. Passing data to subroutine is optional. But for the clarity we need to do it.
2. By default, subroutines adopt pass by reference mechanism. So that; whatever the changes made for the variables inside the subroutine, will be affected in main program (calling program)

|  |  |
| --- | --- |
| Report ZSAMPLE\_SUBROUTINE\_01.  Parameters :  P\_input1 type I,  P\_input2 type I,  P\_input3 type I,  P\_input4 type i.  Data:  L\_output type I,  L\_input type I,  L\_last type i.   * First input   L\_input = p\_input1.  While p\_input1 >= 0.  L\_last = p\_input1 mod 10.  p\_input1 = p\_input1 div 10.  l\_output = l\_output + l\_last.  Endwhile.  Write : ‘Single digit value of input ’, l\_input, ‘ is’,l\_output   * Second input   L\_input = p\_input2.  While p\_input2 >= 0.  L\_last = p\_input2 mod 10.  p\_input2 = p\_input2 div 10.  l\_output = l\_output + l\_last.  Endwhile.  Write : ‘Single digit value of input ’, l\_input, ‘ is’,l\_output   * Third input   L\_input = p\_input3.  While p\_input3 >= 0.  L\_last = p\_input3 mod 10.  p\_input3 = p\_input3 div 10.  l\_output = l\_output + l\_last.  Endwhile.  Write : ‘Single digit value of input ’, l\_input, ‘ is’,l\_output   * Fourth input   L\_input = p\_input4.  While p\_input4 >= 0.  L\_last = p\_input4 mod 10.  p\_input4 = p\_input4 div 10.  l\_output = l\_output + l\_last.  Endwhile.  Write : ‘Single digit value of input ’, l\_input, ‘ is’,l\_output | Report ZSAMPLE\_SUB\_ROUTINE\_02.  Parameters :  P\_input1 type I,  P\_input2 type I,  P\_input3 type I,  P\_input4 type i.  Data:  L\_output type I,  L\_input type I,   * First input   L\_input = p\_input1.  Perform calculate\_sum using p\_input1 changing l\_output.  Write : ‘Single digit value of input ’, l\_input, ‘ is’,l\_output  L\_input = p\_input2.  Perform calculate\_sum using p\_input1 changing l\_output.  Write : ‘Single digit value of input ’, l\_input, ‘ is’,l\_output  L\_input = p\_input3.  Perform calculate\_sum using p\_input1 changing l\_output.  Write : ‘Single digit value of input ’, l\_input, ‘ is’,l\_output  L\_input = p\_input4.  Perform calculate\_sum using p\_input1 changing l\_output.  Write : ‘Single digit value of input ’, l\_input, ‘ is’,l\_output  Form calculate\_sum using p\_in changing p\_out.  Data l\_last type i.  While p\_in >= 0.  L\_last = p\_in mod 10.  p\_in = p\_in div 10.  P\_out = p\_out + l\_last.  Endwhile.  Endform. |

|  |  |
| --- | --- |
| DO 5 TIMES.  Write / sy-index.  ENDDO. |  |