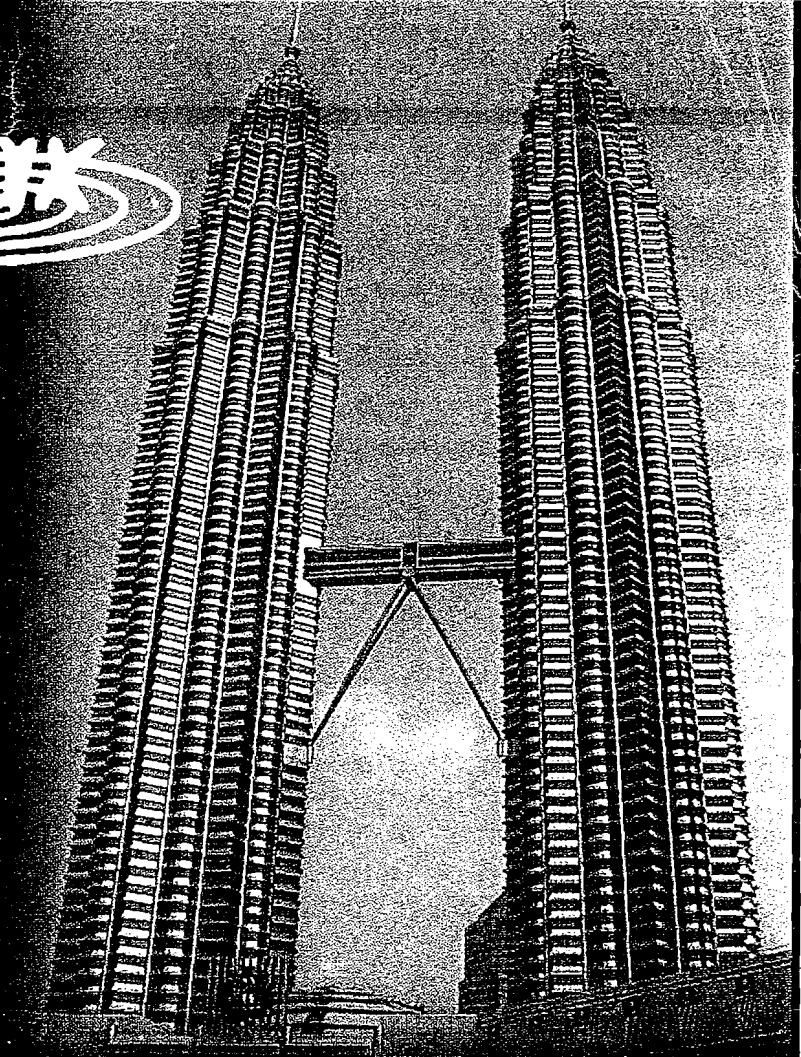


SAP @ e 

**Helps to Reach
The Heights
in SAP**



ABAP/4

The Complete Reference

VOLUME - I

Ganapati Adimulam

INDEX

1.	Introduction to SAP	06
2.	ABAP Data Dictionary	60
3.	Introduction to ABAP Programming	19
4.	Control structures	12
5.	Strings	6
6.	Internal Tables	23
7.	Open SQL	34
8.	ABAP Debugging	13
9.	Reports	74
10.	Modularizing Techniques	44

1.INTRUCTION TO SAP

Duration – 1 Day (* 2 Hrs)

I. WHAT IS SAP R/2?

II. PURPOSE OF R/3?

III. SAP R/3 INDUSTRY SOLUTIONS

IV. SAP R/3 ARCHITECTURE

Introduction to SAP

SAP was founded in 1972 in Walldorf, Germany. It stands for **Systems, Applications and Products in Data Processing**. Over the years, it has grown and evolved to become the world premier provider of client/server business solutions for which it is so well known today. The SAP R/3 enterprise application suite for open client/server systems has established a new standards for providing business information management solutions.

The main advantage of using SAP as your company ERP system is that SAP has a very high level of integration among its individual applications, which guarantee consistency of data throughout the system and the company itself.

What is SAP R/2 ?

SAP R/2

R/2 is SAP AG mainframe solution and was the first compact software package for the whole spectrum of business applications. SAP R/2 runs on mainframes, such as IBM, Siemens, Amdahl. The current version of R/2 is 6.1. This mainframe solution is not open, although with the help of ALE (Application Link Enabled) technology, R/2 can be linked to R/3 systems and share online data.

R/2 is a set of coordinated business applications from SAP, a German company that introduced the product in 1979. R/2 gained popularity until the mid-1990s, when it was superseded by the more capable R/3 product, later updated by mySAP.com. To some extent, R/2 is still in use.

Now more than 20 years old, R/2 continues to be supported by SAP, although support is expected to decline. Using Application Link Enabled (**ALE**) technology, R/2 systems can share data with R/3 and mySAP.com-equipped systems. However, SAP says that it may more cost-effective to migrate to R/3 rather than to stay with R/2, because of the improved support and expanded features available with the current product.

SAP R/2 is the name given to SAP's first ERP solution that was designed for the mainframe. It comprises of a number of modules: RS (Basis), RF (Financial Accounting), RA (Asset Accounting), RK (Cost Accounting), RK-P (Project Costing), RM-INST (Plant Maintenance), RM-MAT (materials Management), RM-PPS (Production Planning and Control), RM-QSS (Quality Assurance), RP (Human Resources) and RV (Sales and Distribution).

What is the Purpose of R/3?

The sole purpose of an R/3 system is to provide a suite of tightly integrated, large-scale business applications.

The standard set of applications delivered with each R/3 system are the following :

- PP (Production Planning)
- MM (Materials Management)
- SD (Sales and Distribution)
- FI (Financial Accounting)
- CO (Controlling)
- AM (Fixed Assets Management)
- PS (Project System)
- WF (Workflow)
- IS (Industry Solutions)
- HR (Human Resources)
- PM (Plant Maintenance)
- QM (Quality Management)

These applications are called the functional areas, or application areas, or at times the functional modules of R/3. All of these terms are synonymous with each other.

Traditionally, businesses assemble a suite of data processing applications by evaluating individual products and buying these separate products from multiple software vendors. Interfaces are then needed between them. For example, the materials management system will need links to the sales and distribution and to the financial systems, and the workflow system will need a feed from the HR system. A significant amount of IS time and money is spent in the implementation and maintenance of these interfaces.

R/3 comes prepackaged with the core business applications needed by most large corporations. These applications coexist in one homogenous environment. They are designed from the ground up to run using a single database and one (very large) set of tables. Current production database sizes range from 12 gigabytes to near 3 terabytes. Around 8,000 database tables are shipped with the standard delivery R/3 product.

SAP R/3 Overview

SAP R/3 is SAP's integrated software solution for **client/server and distributed open systems**. SAP's R/3 is the world's most-used standard business software for client/server computing. R/3 meets the needs of a customer from the small grocer with 3 users to the multi-billion dollar companies **The software is highly**

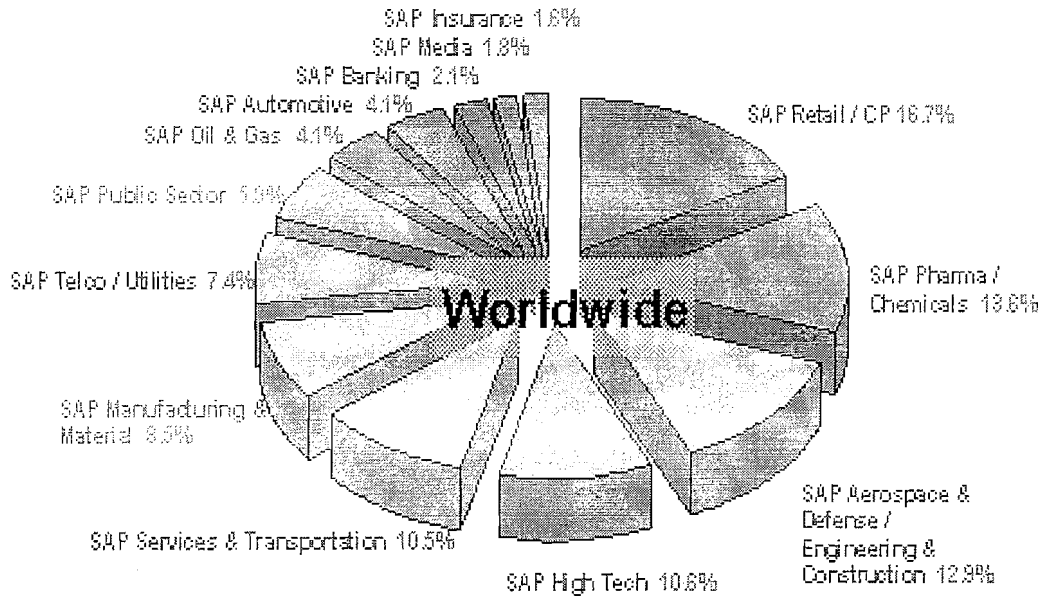
customizable using SAP's proprietary programming language, ABAP/4. R/3 is scalable and highly suited for many types and sizes of organizations.

The R/3 architecture is comprised of application and database servers. The application servers house the software and the database servers handle document updates and master file databases. The system can support an unlimited number of servers and a variety of hardware configurations. For more info. see SAP R/3 Architecture at SAP home page.

SAP R/3 is based on various hardware and software architectures, running on most types of UNIX, on Windows NT and OS/400

SAP R/3 runs on several databases Oracle, Adabas D, Informix, DB2 for UNIX, DB2/400, Microsoft's SQL Server 6.0.

SAP R/3 Installations by Industry *



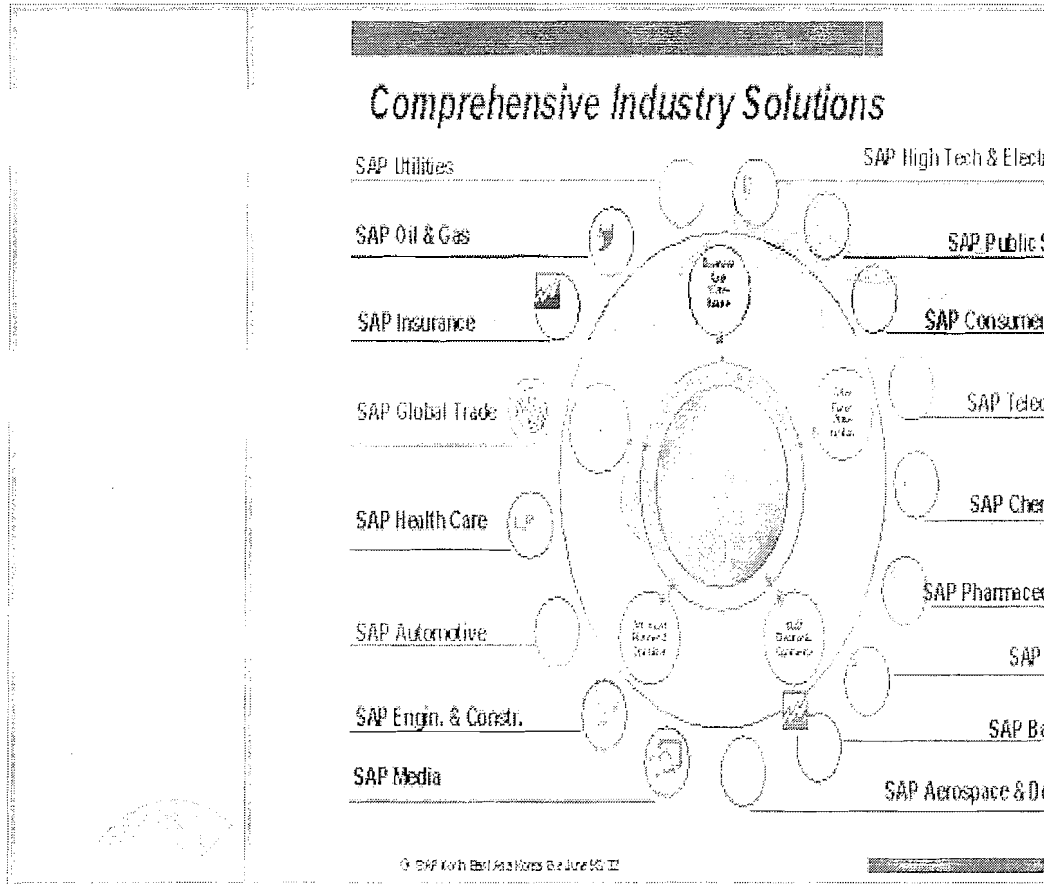
*Customer Installations only

As of December, 31st, '97



SAP R/3 Industry Solutions

R/3 understands industry and hence offers comprehensive functionality for all standard business needs in any enterprise. With integrated industry-specific business processes, R/3 meets the individual requirements of numerous industries.



SAP R/3 Architecture :

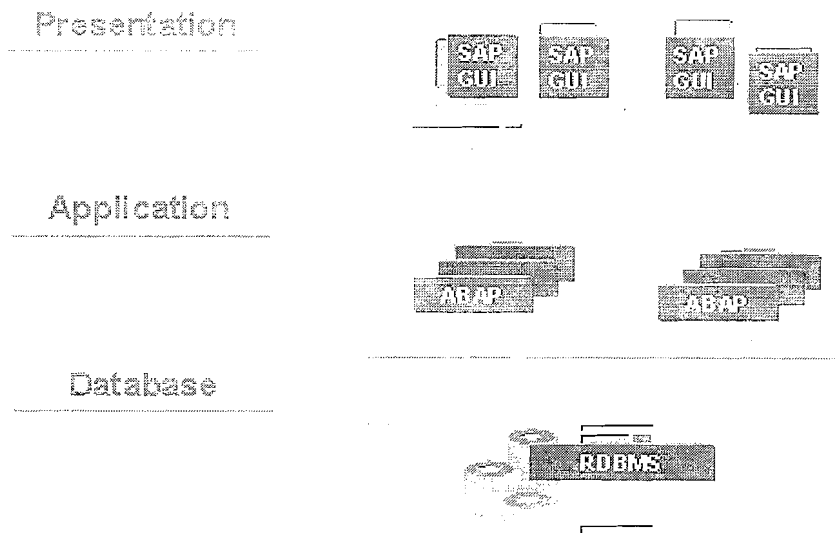
Definition

The R/3 System has a specific architecture with a specific set of logical services.

Structure

In the three-tier, client/server architecture of the R/3 System, there are three service layers:

R/3 System Service Layers



IDES – the "Internet Demonstration and Evaluation System" in the R/3 System, represents a model company. It consists of an international group with subsidiaries in several countries. IDES contains application data for various business scenarios that can be run in the SAP System. The business processes in the IDES system are designed to reflect real-life business requirements, and have access to many realistic characteristics. IDES uses easy-to-follow business scenarios to show you the comprehensive functions of the R/3 System. The focal point of IDES, however, is not the functionality itself, but the business processes and their integration.

These IDES business processes are described in detail within this online documentation. The individual demos provide you with an overview of the master data, and contain step-by-step instructions of how to execute the individual processes.

IDES not only covers the Logistics area, but also Financials, and Human Resources. It demonstrates how the R/3 System is able to support practically all types of industries, from discrete production through to process industries, from engineering-to-order to repetitive manufacturing. However, IDES is not a sector-oriented model company. The individual processes are based on practice-oriented data for sectors such as Retailing or is managed by SAP just as any regular business enterprise. SAP regularly updates the IDES data (master data, transaction data, and customizing). Banking. The IDES group manufactures products as diverse as elevators, motorcycles, and paints.

IDES We also carry out period-end closing and plan with different time-horizons. Transaction data are generated to ensure that the information systems in all areas have access to realistic evaluation data. We are constantly implementing new, interesting business scenarios to highlight the very latest functions available in the R/3 System. New functions are represented and documented by IDES scenarios.

Above all, IDES shows you the possibilities of the integrated applications in the SAP System. It Covers all aspects of a business enterprise, including Human Resources, Financial Accounting, Product Cost Planning, Overhead Management, Profitability Analysis, Planning, Sales and Distribution, Materials Management, Production, and much, much more.

IDES shows you how the R/3 System supports production processes, the supply chain, and the efficient usage of global resources. Or perhaps you would like to increase your understanding of just-in-time-production or the integration of the electronic KANBAN system in an MRP II environment? IDES provides the ideal way to learn about areas such as Product Cost Controlling, Activity-Based Costing, or integrated Service Management and Plant Maintenance. How to manage high inflation is just one of the ever-growing number of IDES business scenarios that you can choose from.

2.ABAP Dictionary

- a. Purpose
- b. Types of Tables
- c. Technical Requirements to create the table
- d. Data Dictionary Data Types
- e. Domain
- f. Data Element
- g. Steps to create the table
- h. Structures
- i. Foreign key Relationship
- j. Views
- k. Lock Objects

ABAP Dictionary:-

The ABAP Dictionary centrally describes and manages all the data definitions used in the system. The ABAP Dictionary is completely integrated in the ABAP Workbench. All the other components of the Workbench can actively access the definitions stored in the ABAP Dictionary.

The ABAP Dictionary supports the definition of user-defined types (data elements, structures and table types). You can also define the structure of database objects (tables, indexes and views) in the ABAP Dictionary. These objects can be automatically created in the database with this definition.

The most important object types in the ABAP Dictionary are **tables, views, types (data elements, structures, and table types), domains, search helps and lock objects.**

Purpose

Data definitions (metadata) are created and managed in the ABAP Dictionary. The ABAP Dictionary permits a central description of all the data used in the system without redundancies. New or modified information is automatically provided for all the system components. This ensures data integrity, data consistency and data security.

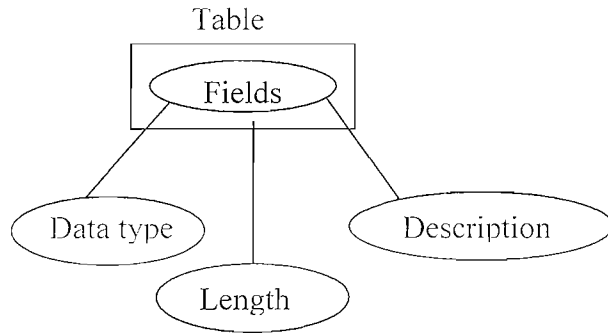
Tables are defined in the ABAP Dictionary independently of the database. A table having the same structure is then created from this table definition in the underlying database.

A table definition in the ABAP Dictionary contains the following components:

- **Table fields** define the field names and data types of the fields contained in the table
- **Foreign keys** define the relationships between the tables.
- **Technical settings** control how the table should be created in the database.

Steps to create the table in DDIC:

Note: A Table is Simply Group of fields and which can store multiple records.



1) **Create the Structure of the table i.e the list of Fields and their Technical attributes such as Data types, Lengths and Descriptions.**

2) **Allocate the Memory for the above table structure in Database.**

Technical Terms that helps in Creation of Data Base Table:-

Delivery Class:-

It Defines the type of the data that is going to be stored and also it defines the owner of the Table and also the delivery class controls the transport of table data for installation, upgrade, client copy and when transporting between customer systems.

<u>Delivery Class</u>	<u>Description</u>
A	Application table (master and transaction data)
C	Customizing table, maintenance only by cust., not SAP import
L	Table for storing temporary data, delivered empty
G	Customizing table, protected against SAP Upd., only INS all.
E	Control table, SAP and customer have separate key areas
S	System table, maint. only by SAP, change = modification
W	System table, contents transportable via separate TR objects

Data Class: - It is the physical area or table space that the table has to be stored in the Data Base.

<u>Data Class</u>	<u>Description</u>
APPL0	Master data, transparent tables
APPL1	Transaction data, transparent tables
APPL2	Organization and customizing
USER	Customer Data Class
USER1	Customer Data Class

Size Category:- The size category defines the expected space required for the table in the database. You can choose a size category from 0 to 4 for your table. Each category is assigned a certain fixed memory size in the database, which depends on the database system used.

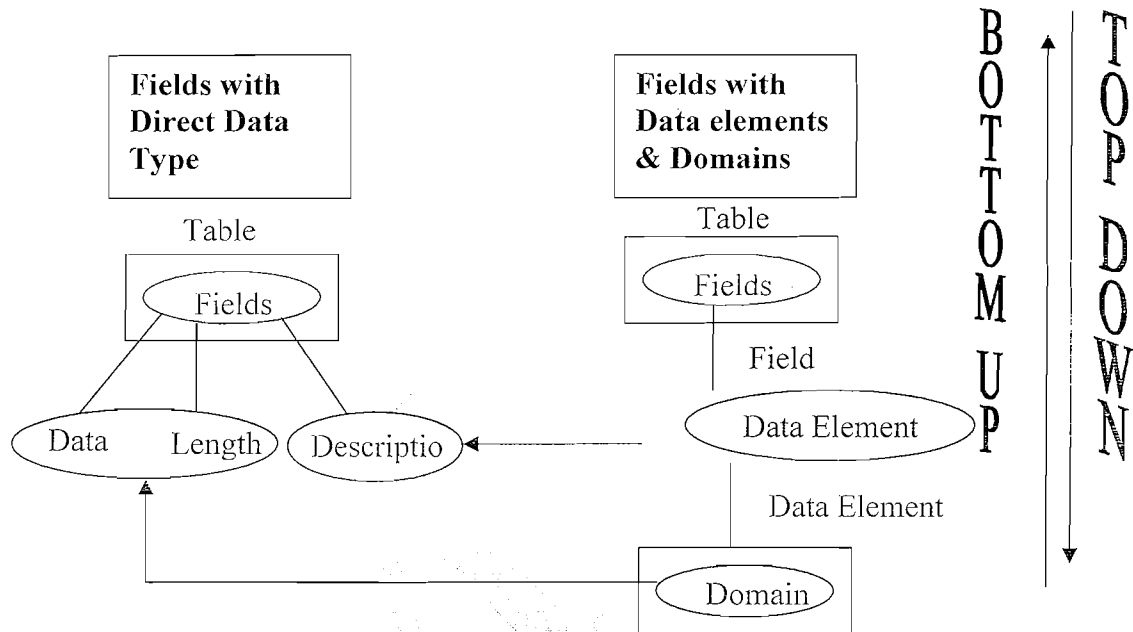
<u>Size Category</u>	<u>Number of data records of table expected</u>
0	0 to 2.600
1	2.600 to 10.000
2	10.000 to 42.000
3	42.000 to 170.000
4	170.000 to 13.000.000

Table Maintenance Allowed:

If you click on this check box, you can enter the entries into the table.

Table Creation in DDIC:

Note: While Creating the Table, We Can Provide the technical attributes such as Data type, Length and Description of the Fields in 2-ways (Refer the below diagram).



- a) One Way is providing the technical attributes Directly.
I.e Provide the Data type, Length and Description Individually.
- b) 2nd Way is providing the technical attributes through Data Elements and Domains.
i.e Provide the Data type, Length and Description **by grouping into Data Elements and Domains.**

Note : We Never Provide the Technical attributes of the field using Direct type instead it is always through Domains and Data Elements.

Domain:- It is used to define the Technical Attributes (Data Type and Length) and the Value Ranges (Fixed Values and Intervals). A domain is assigned to a data element. All table fields or structure components that use this data element then have the value range defined by the domain.

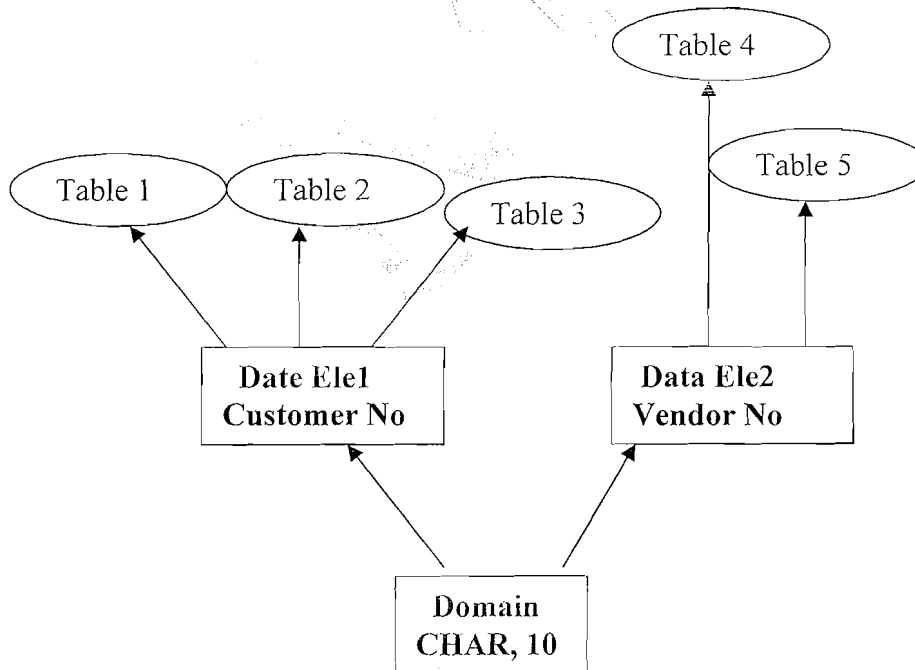
Data Element:- A Data Element describes either an elementary type or a reference type. An elementary type is defined by the built-in data type, length and possibly the number of decimal places. These type attributes can either be defined directly in the Data Element or copied from a domain.

Information about the meaning (description) of a table field or structure component and information about editing the corresponding screen field can be assigned to a Data Element. This information is automatically available to all screen fields that refer to the Data Element.

Data Elements are used to provide F1 Help.

Advantages of Domains and Data Elements:

a) Re-usability



- b) **Foreign Key Relationships** Can be maintained Because the Foreign Key Field and Check table field should have the Same Domain.

Data Dictionary Data Types Helps to Create the Domains:

Data Types Descriptions

ACCP	Posting period YYYYMM
CHAR	Character string
CLNT	Client
CUKY	Currency key, referenced by CURR fields
CURR	Currency field, stored as DEC
DATS	Date field (YYYYMMDD) stored as char(8)
DEC	Counter or amount field with comma and sign
FLTP	Floating point number, accurate to 8 bytes
INT1	1-byte integer, integer number <= 255
INT2	2-byte integer, only for length field before LCHR or LRAW
INT4	4-byte integer, integer number with sign
LANG	Language key
LCHR	Long character string, requires preceding INT2 field
LRAW	Long byte string, requires preceding INT2 field
NUMC	Character string with only digits
PREC	Precision of a QUAN field
QUAN	Quantity field, points to a unit field with format UNIT
RAW	Uninterrupted sequence of bytes
TIMS	Time field (hhmmss), stored as char(6)
VARC	Long character string, no longer supported from Rel. 3.0
STRING	Character string of variable length
RAWSTRING	Byte string of variable length
UNIT	Unit key for QUAN fields

Requirement:

Create the table to Maintain the Vendor General Data using the Below Information.

Filed Name	Data Type	Length	Description
LIFNR	CHAR	10	Vendor Account No
NAME1	CHAR	35	Name Of the Vendor
ORT01	CHAR	35	City
ORT02	CHAR	35	District
STRAS	CHAR	35	House Number & Street
LAND1	CHAR	3	Country Key

Domains and Data Elements Required to Create for the Above Table :

Data Elements:

Since all the fields LIFNR, NAME1, ORT01, ORT02, STRAS, LAND1 require Unique Descriptions we need unique Data Elements for each Field.

Filed Name	Data Element to Be Created
LIFNR	ZGLIFNR(Vendor Account No)
NAME1	ZGNAME1(Name Of the Vendor)
ORT01	ZGORT01(City)
ORT02	ZGORT02(District)
STRAS	ZGSTRAS(House Number & Street)
LAND1	ZGLAND1(Country Key)

DOMAINS: Since all the Below Fields have the same Technical Characteristics (CHAR, 35), One Domain is enough with the Data Type CHAR and Length 35.

NAME1 CHAR35
 ORT01 CHAR35
 ORT02 CHAR35
 STRAS CHAR35

Filed Name	Domain names to be Created
NAME1	ZGCHAR35 , Which is Common for all the Fields
ORT01	
ORT02	
STRAS	
LIFNR	ZGCHAR10(CHAR,10)
LAND1	ZGCHAR3(CHAR,3)

Now We are ready with all the Domains and Data Elements Information so that we can start creating the table.

Create the Table Using the Following Domains & Data Elements:

Field Name	Data Element	Domain
LIFNR	ZGLIFNR	ZGCHAR10
NAME1	ZGNAME1	ZGCHAR35
ORT01	ZGORT01	ZGCHAR35
ORT02	ZGORT02	ZGCHAR35
STRAS	ZGSTRAS	ZGCHAR35
LAND1	ZGLAND1	ZGCHAR3

Note1:

We will create all the required Domains and Data Elements first so that we can simply provide the same while creating the Table.

Note 2: We Can Create the Data Elements in Two Ways

i.e. Create the Domain -> Data Element(BOTTOM- UP)

or

Start with Data Element Creation then Domain from there.

(TOP - DOWN)

Steps to Create Domain: (BOTTOM- UP Approach)

Execute SE11 and select the '**Domain**' option.

Provide the name 'ZGCHAR10' and click on '**Create**'.

Here enter the '**Short Text**' and enter the '**Data type , Number of characters**' and Press '**ENTER**'.

Domain ZGCHAR10 Active
 Short text char,10

Attributes Definition Value range

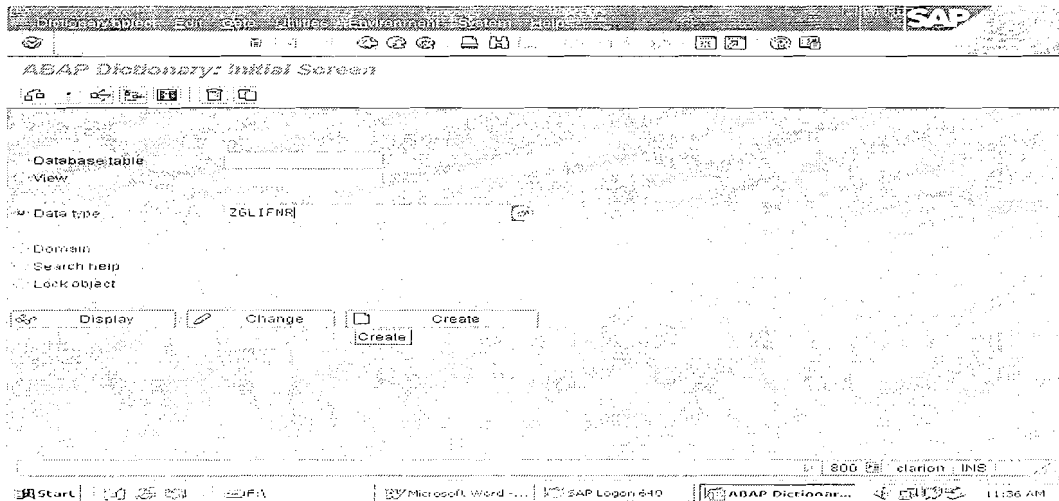
Formatting		
Data type	CHAR	Character string
No. characters	10	
Decimal places	0	
Output characteristics		
Output length	10	
Convers. routine		
<input type="checkbox"/> Sign		
<input type="checkbox"/> Lowercase		

Save, Check and Active.
 Make Sure that the Domain is Activated.

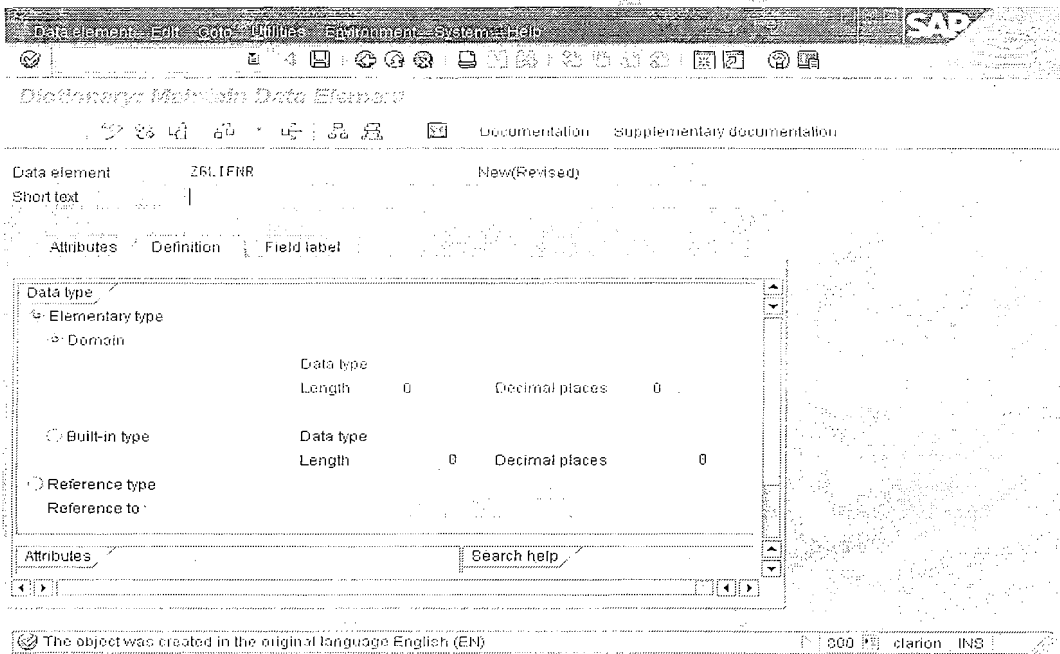
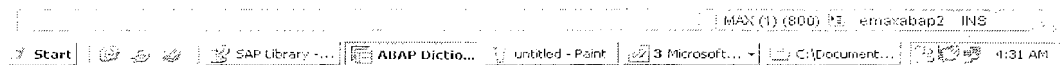
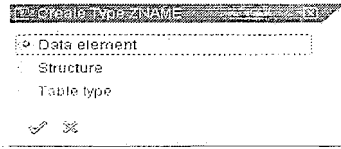
Steps to Creating Data Element.

Execute 'SE11'.

Select the radio Button 'Data Type' and Enter the name of the Data Type(Data Element).



Select the Radio Button Data Element press 'Enter'.



Enter the 'Short Text and the Domain which is already created ZGCHAR10. Press ENTER After Domain is entered . So that it Copies the technical attributes from Domain to Data element.. Save, Check and Active the Data Element..

Data element ZGLIFNR Active
 Short text Vendor Number

Attributes Definition Field label

Data type

Elementary type

Domain Z6CHAR10 Char,10

Data type CHAR

Length 10 Decimal places 0

Built-in type

Data type

Length 0 Decimal places

Reference type

Reference to

Maintain Field Label.

This Label will be displayed when we refer the same While Designing the Screen in Module Pool Programming.

Dictionary: Maintain Data Element

Documentation Supplementary documents

Data element ZGLIFNR Active
 Short text Vendor Account Number

Attributes Definition Field label

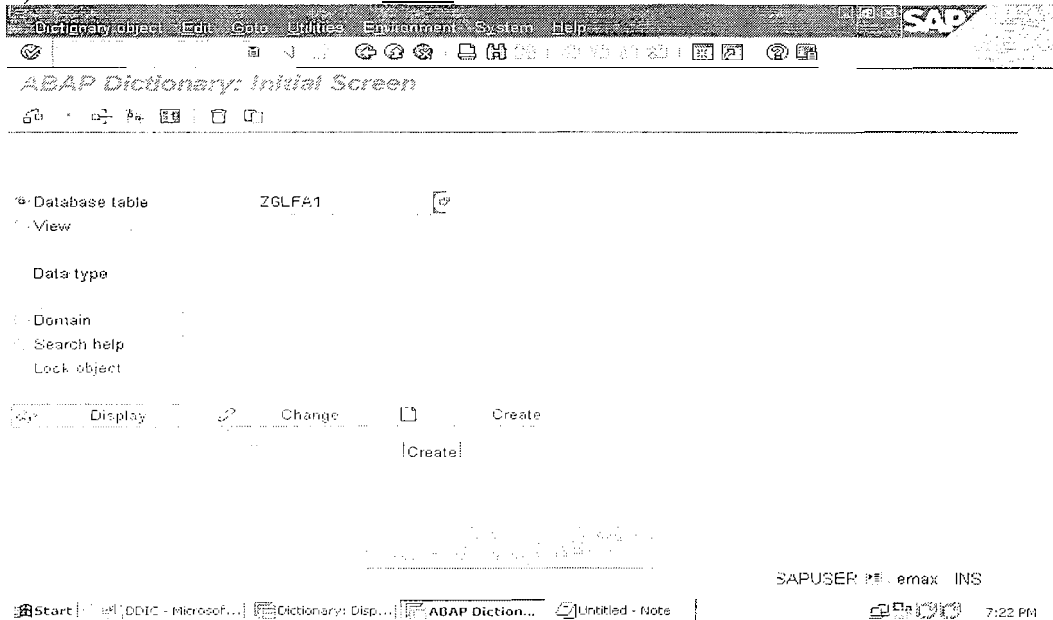
Length Field label

Short
 Medium
 Long
 Heading 25 Vendor Account Number

Note :
 Repeat the Same Procedure to Create all the Domains and attach the Domains to the Corresponding Data Elements while Creating the Data Elements.

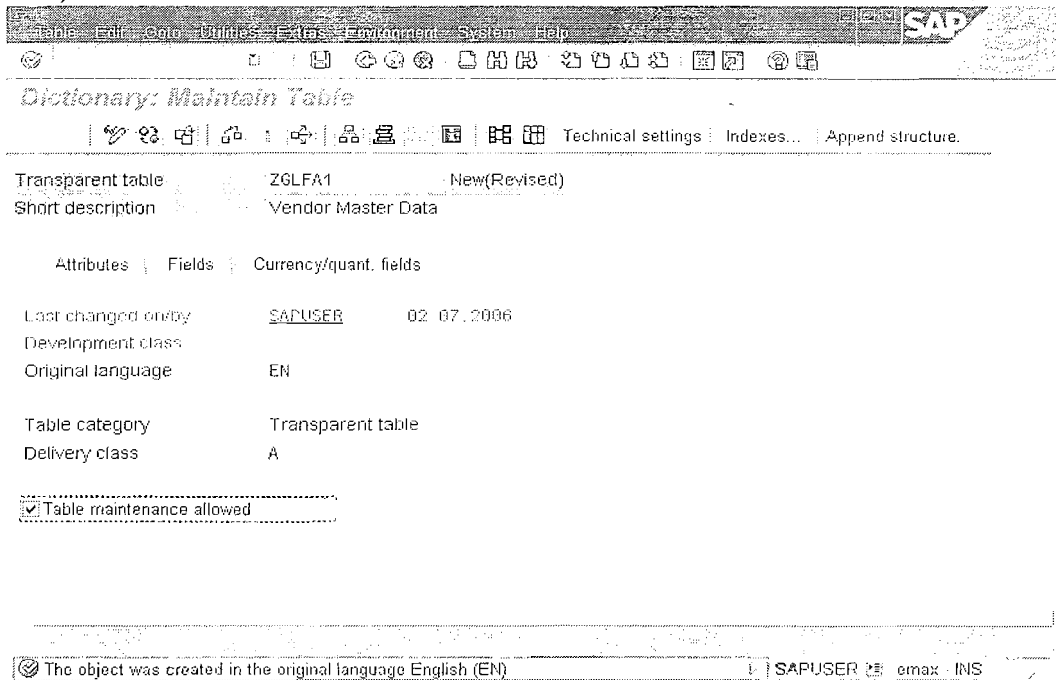
Step by Step Procedure to Create the Table :

1) Execute Transaction **SE11**. The initial screen is as follows :

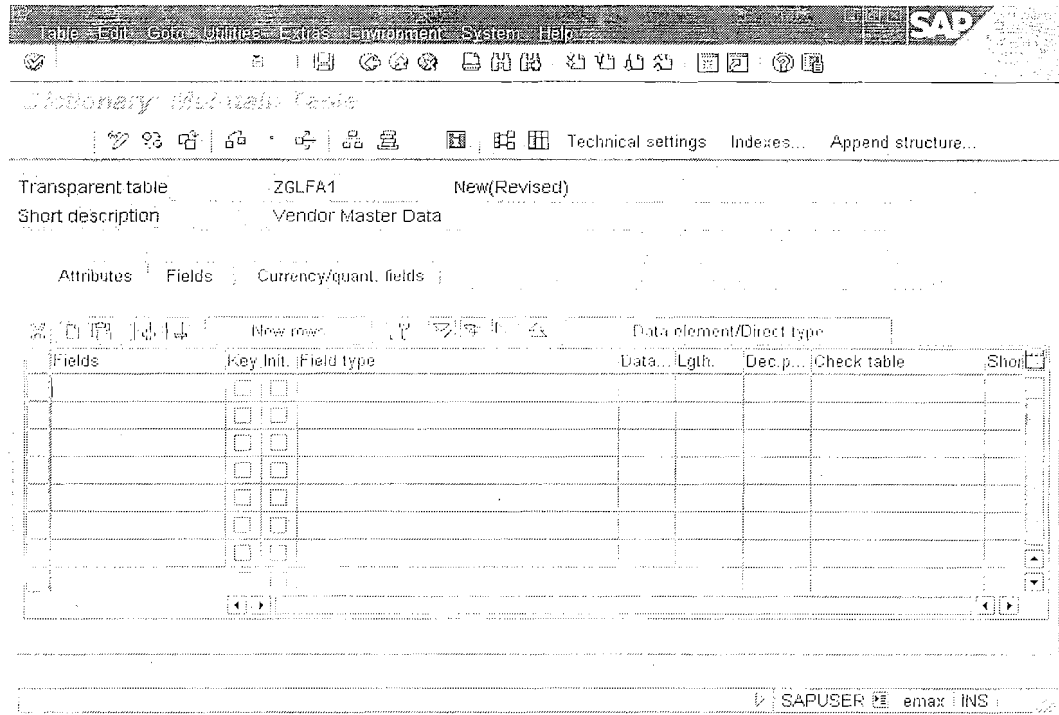


2) Click on the radio-button '**Database Table**' as shown above, enter the database name as '**ZGLFA1**' to be created and click on '**CREATE**' button.

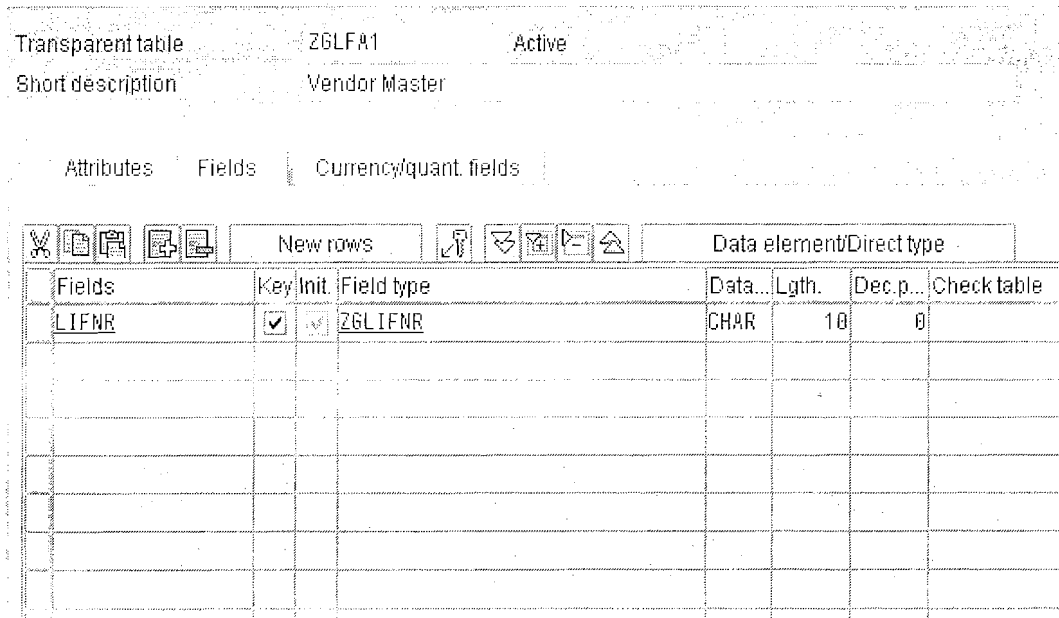
Enter the fields which are mandatory like '**Short Description**', '**Delivery class as 'A'**', and '**Check Table Maintenance Check Box**' in the screen as shown below.



Then click on the **Fields** tab. Following screen appears.



3) Provide the Field name and the Data Element for LIFNR i.e ZGLIFNR, which is already created. Press Enter after entering the Data Element so that we get all the information attached to it (Data Type, Length, Description).



Domain Z6CHAR35 Active
 Short text char 35

Attributes Definition Value range

Formatting

Data type CHAR Character string
 No. characters 35
 Decimal places 0

Output characteristics

Output length 35
 Convers. routine
 Sign
 Inverse case

Provide the Data type and Length.

Save , Check and Active.

Press **F3(BACK)**

Note: Make sure that the Domain is Activated Before going back.

Note: After F3 we are at Data Element screen. Save, Check, Activate the Data Element.

Data element Z6NAME1 Active
 Short text Name of the vendor

Attributes Definition Field label

Data type

Elementary type

Domain Z6CHAR35 char 35

Data type CHAR
 Length 35 Decimal places 0

Built-in type

Data type
 Length 0 Decimal places 0

Reference type

Reference to

Save , Check and Activate.(Now the Data element Z6NAME1 is activated).

Press F3 to go back to the Table Screen.

Note: Observe here the Data Type, length and Description are copied from the Data Element ZGNAME1(because the Domain is already attached to the Data Element).

Transparent table ZGLFA1 Active
 Short description Vendor Master

Attributes Fields Currency/quant. fields

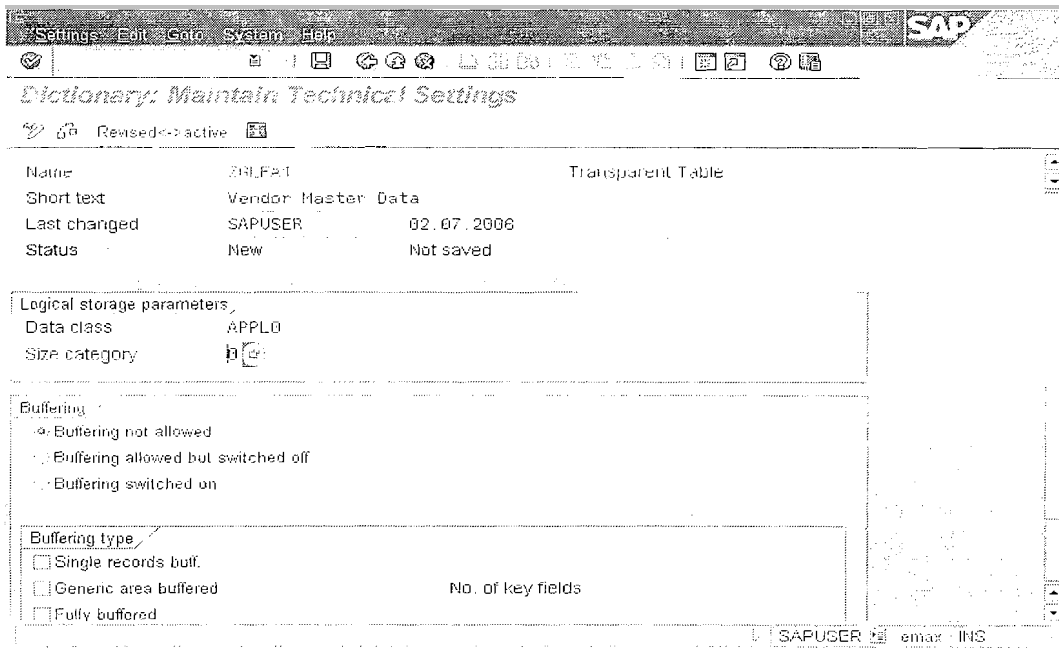
Fields	Key/Init.	Field type	Data...	Lgth.	Dec. p...	Check table	Short text
LIFNR	<input checked="" type="checkbox"/>	ZGLIFNR	CHAR	10	0		Vendor No
NAME1	<input type="checkbox"/>	ZGNAME1	CHAR	35	0		Name

Note 1: Repeat the same procedure for all the fields in the table.

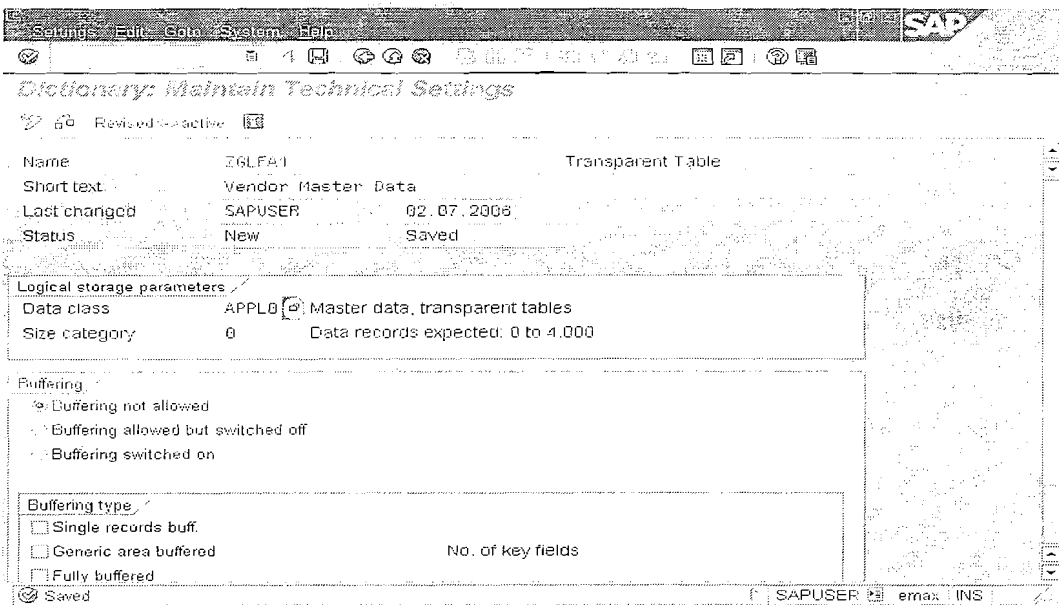
Note 2: For LIFNR Data Element is already created (Bottom-Up).

The screenshot shows the SAP ABAP Data Dictionary interface. At the top, there is a menu bar (Table, Edit, Goto, Utilities, Extras, Environment, System, Help) and a toolbar. Below the menu bar, the text 'Dictionary: Main table' is visible. The main area shows the table details for ZGLFA1 (Vendor Master Data). Below this, there are tabs for 'Attributes', 'Fields', and 'Currency/quant. fields'. The 'Fields' tab is active, showing a table of fields. At the bottom of the screen, the status bar shows 'SAPUSER emax INS'. A callout box with a pointer to the 'Technical Settings' button in the toolbar contains the text: 'To set the Technical settings of the table'.

Now Click on the 'Technical Settings' on the tool bar. A screen is displayed as below.

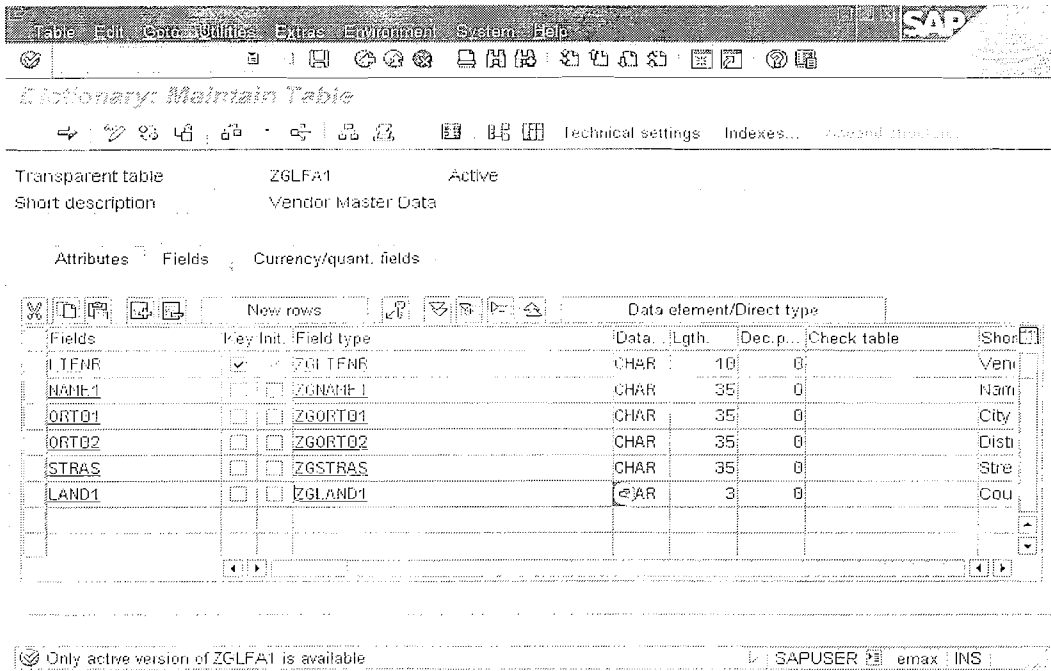


Enter the Data Class and Size Category. Save It.

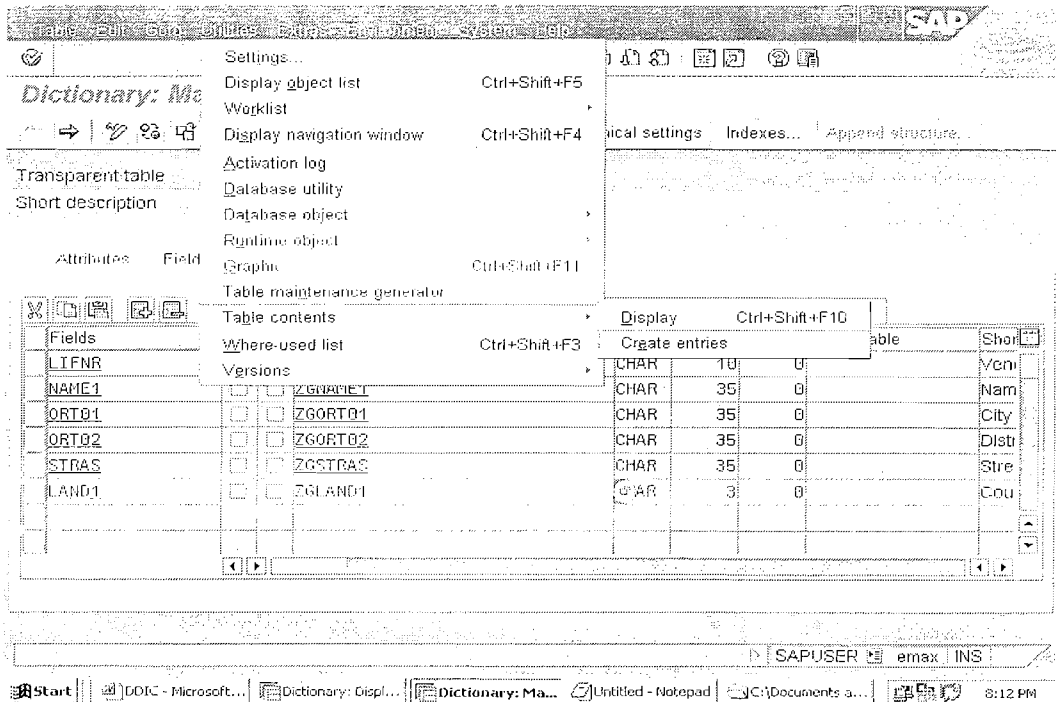


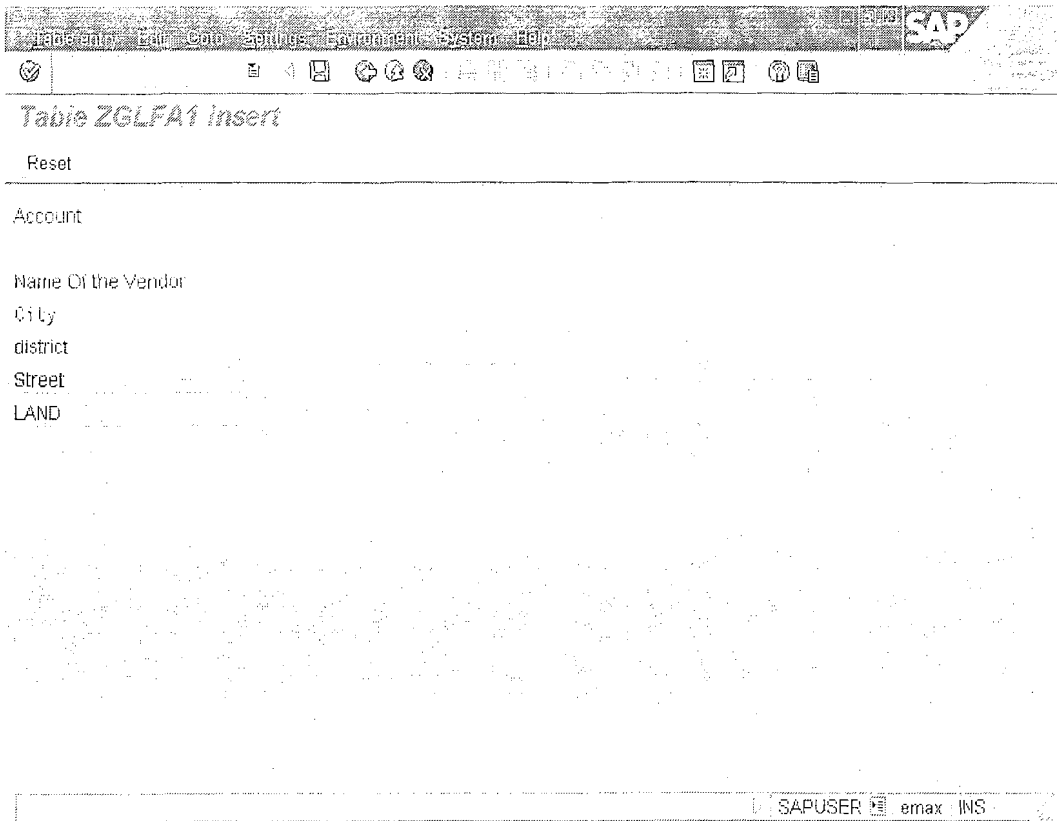
Press F3 to come back to the main screen.

Save, Check, Activate the table.

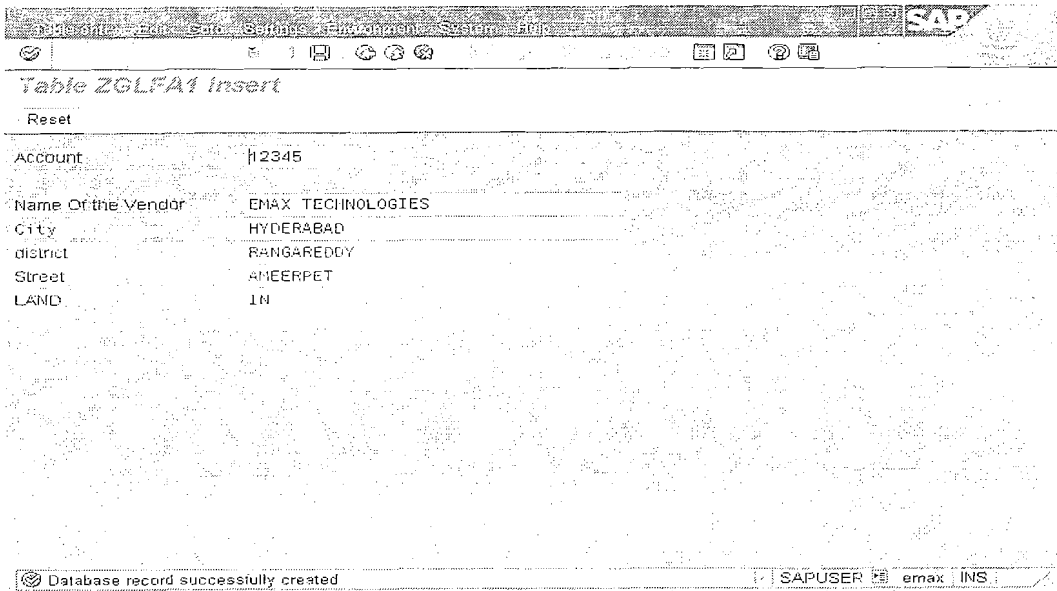


Path to maintain the Data on the Table Utilities -> Table contents -> **Create Entries.**





Above screen is displayed. Enter the Vendors details. After entering save **Ctrl + S** or Press **Save** Button to update the table contents.



Path to view the contents of the table. Utilities -> Table Contents -> **Display**.

Field	Length	Unit	Short Description
LIFNR	10	U	Vendor
NAME1	35	0	Name
ORT01	35	0	City
ORT02	35	0	District
STRAS	35	0	Street
LAND1	3	0	Country

SAPUSER emax INS 8:10 PM

Number of entries

Vendor Account No | | | to

Name Of the Vendor | | | to

City | | | to

District | | | to

Street | | | to

Country Key | | | to

Width of output list 350

Maximum no. of hits 200

SAPUSER emax INS

Press F8 for all the records or provide the Required Vendor Account No's and then Execute(F8).

The screenshot shows the SAP Data Browser interface for table ZGLFA1. The table has 6 columns, with 5 displayed. The entry shown is:

Vendor Account No	Name Of the Vendor	City	Dist
12345	EMAX TECHNOLOGIES	HYDERABAD	RANG

Working With STRUCTURES :-

If we have three Data Base tables ZGLFA1,ZGKNA1,ZGT001

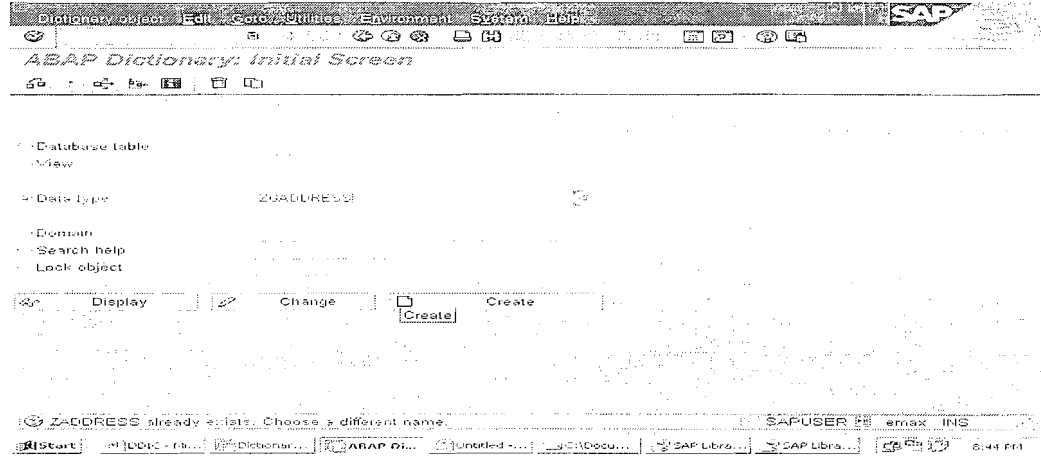
ZGLFA1	ZGKNA1	ZGT001
LIFNR	KUNNR	BUKRS
NAME1	NAME1	NAME1
ORT01	ORT01	ORT01
ORT02	ORT02	ORT02
STRAS	STRAS	STRAS
LAND1	LAND1	LAND1

Note : Here the Same Set of fields NAME1,ORT01,ORT02,STRAS,LAND1 are repeated in all the tables . Instead Of maintaining all the fields in all the above tables it is better to group the repeated fields as a structure and where we can simply include the same structure in any number of Custom(User Defined) tables.

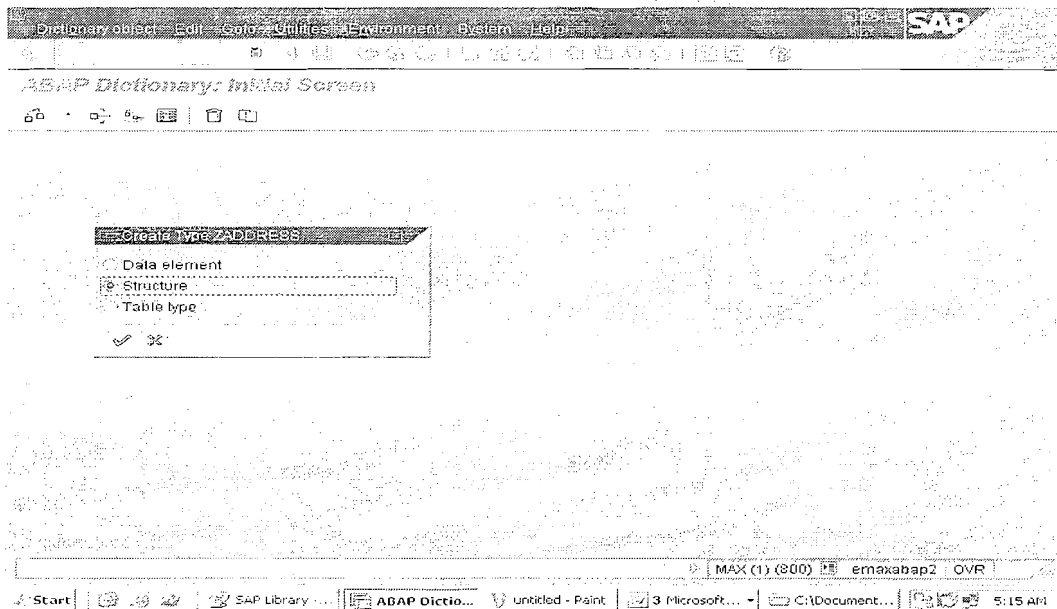
The central definition of structures that are used more than once makes it possible for them to be changed centrally. The active ABAP Dictionary then makes this change wherever required. ABAP programs or screen templates that use a structure are automatically adjusted when the structure is changed (see Runtime Objects). This ensures the greatest possible consistency of the data definition, also for complex programs.

Steps to create Structures.

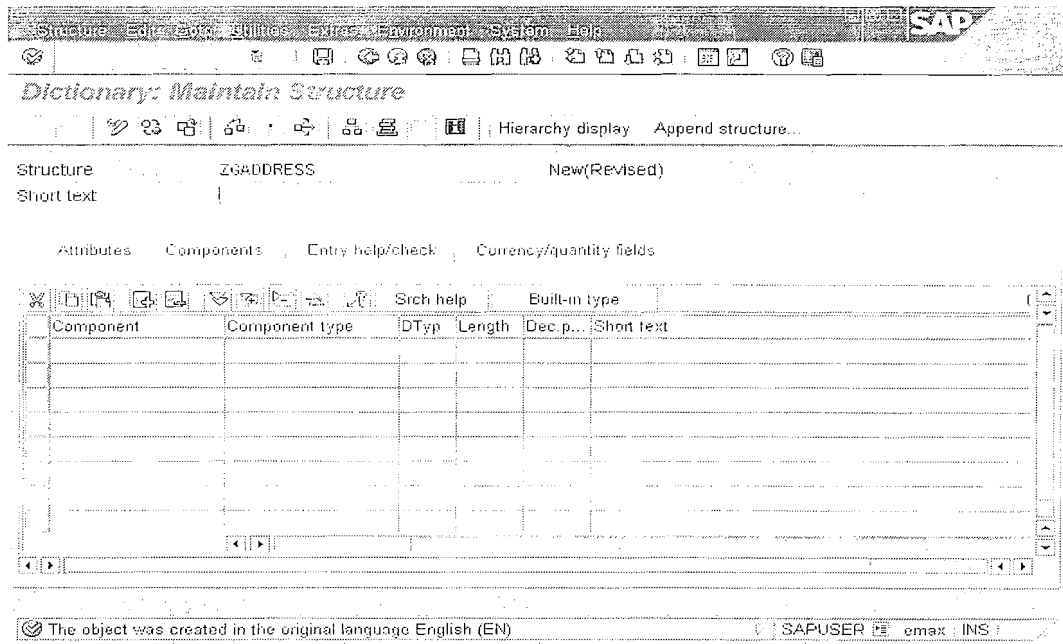
Execute SE11. Click on the 'Data Type' option on the screen. Enter the Name and click on 'Create'.



Select the option 'Structure' in the next screen.



Press Enter.

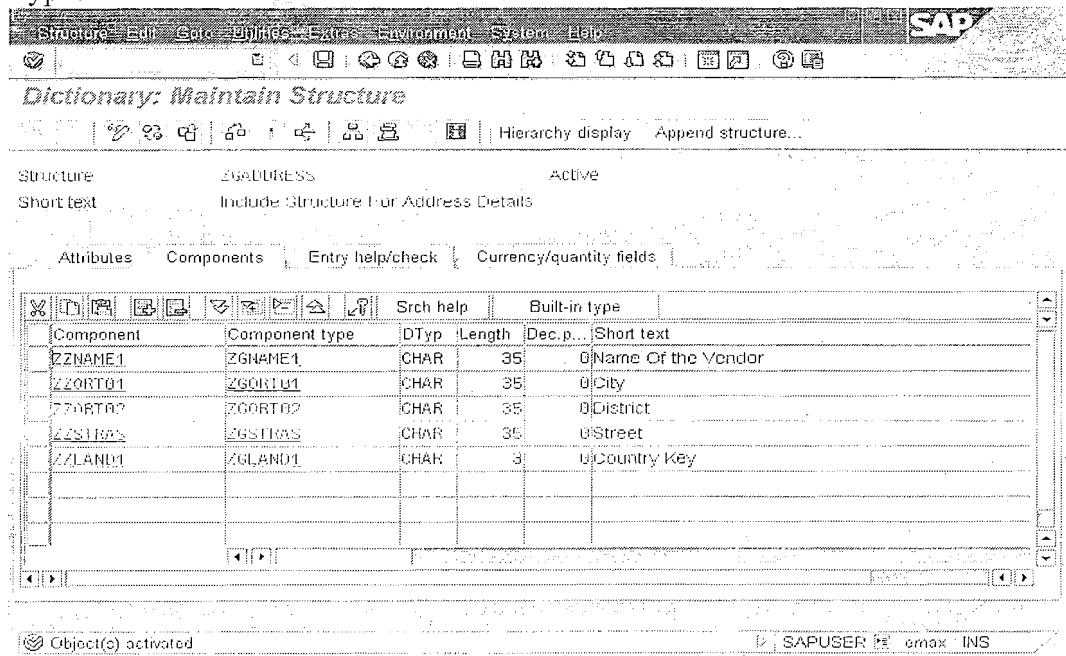


Enter the **Short text**, **Component(Field Name)** and **Component Type(Data Element)**.

NOTE: Here all the component name start with 'ZZ' as per the SAP Recommendation.

Please Make use of the Data Elements which we already created for the Same.

Save, Check, Activate After Providing all the Components and Component Types.



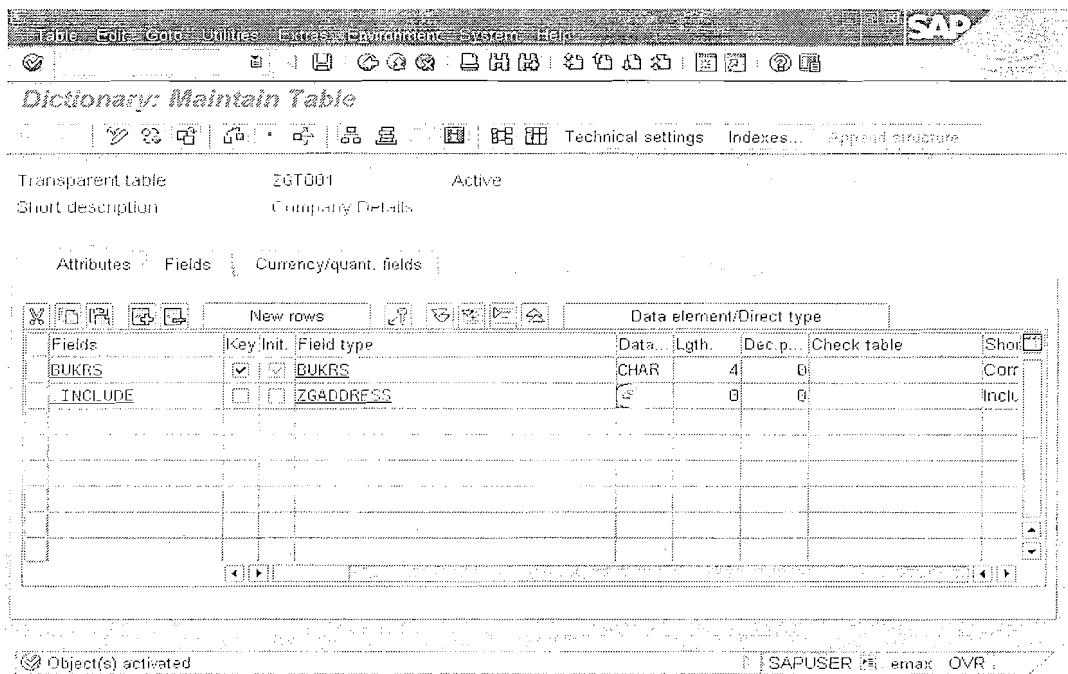
Steps to INCLUDE Structure: in all the Required Tables.


Include Structure is used to include the structure to a custom Table.

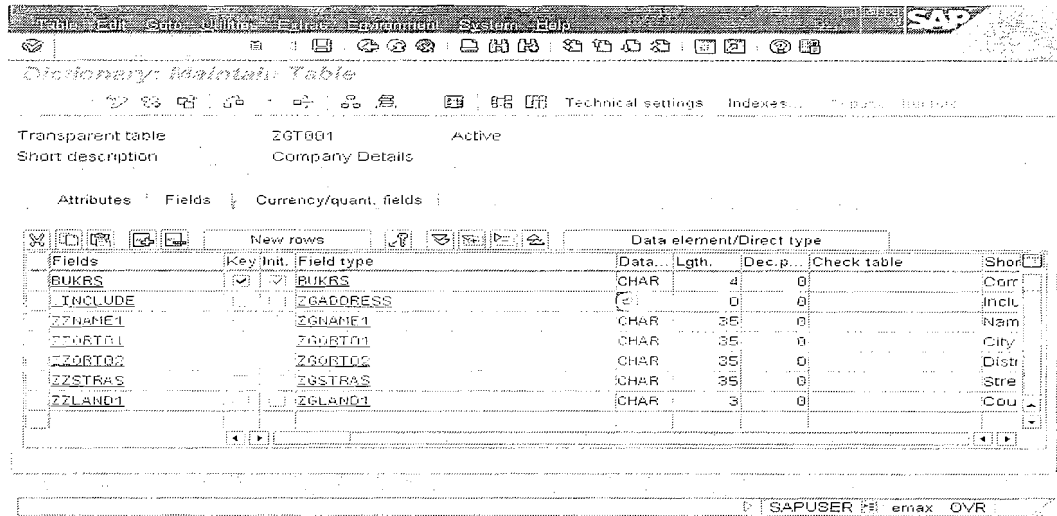
Open the Database Table in Change Mode, i.e Execute **SE11** and Provide the Name of the **Table** and Click on **Change**

Select the Row i.e where you want to **INCLUDE** the Structure and then click on '+' or '**New Rows**' button

Provide **.INCLUDE** as Field name and **Structure name** as Field type as shown below. Save, Check and Activate.



Click on the **Expand All Include**  Button to see the fields included in the Structure.



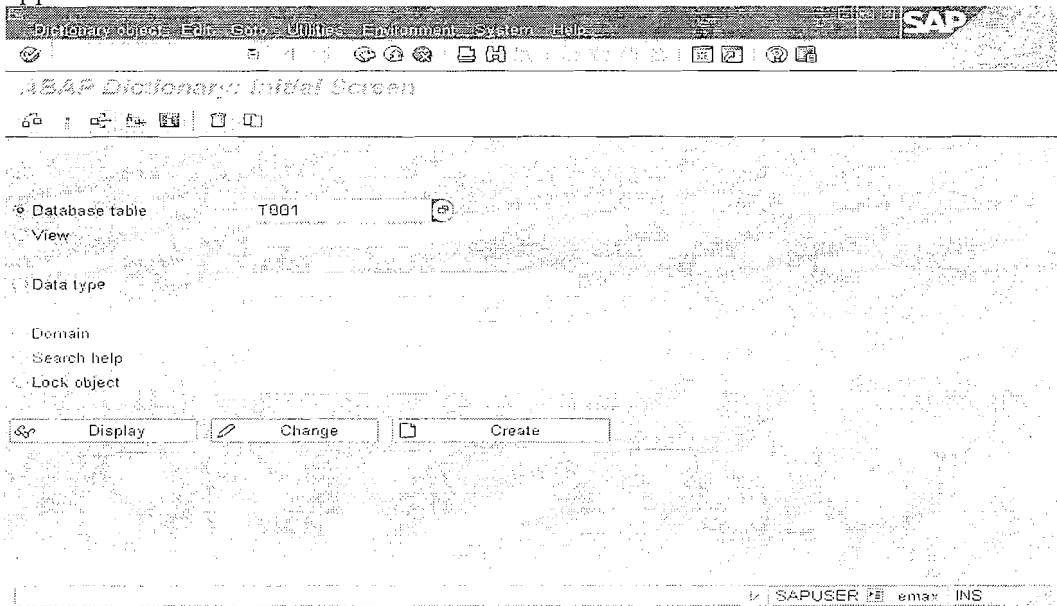
APPEND Structure:-

Here as we use 'INCLUDE' structure for custom table or user defined table, we use '**APPEND**' Structure for adding the required fields for the standard database tables.

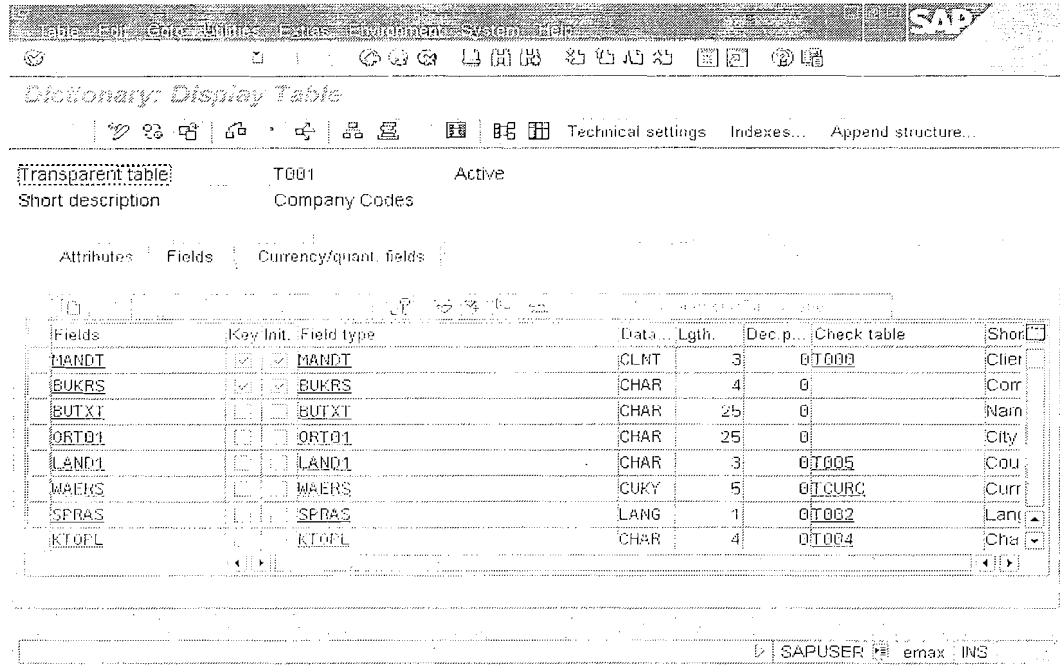
Note : Even adding one field to the standard table also should be done via APPEND Structure

Procedure to APPEND structure:-

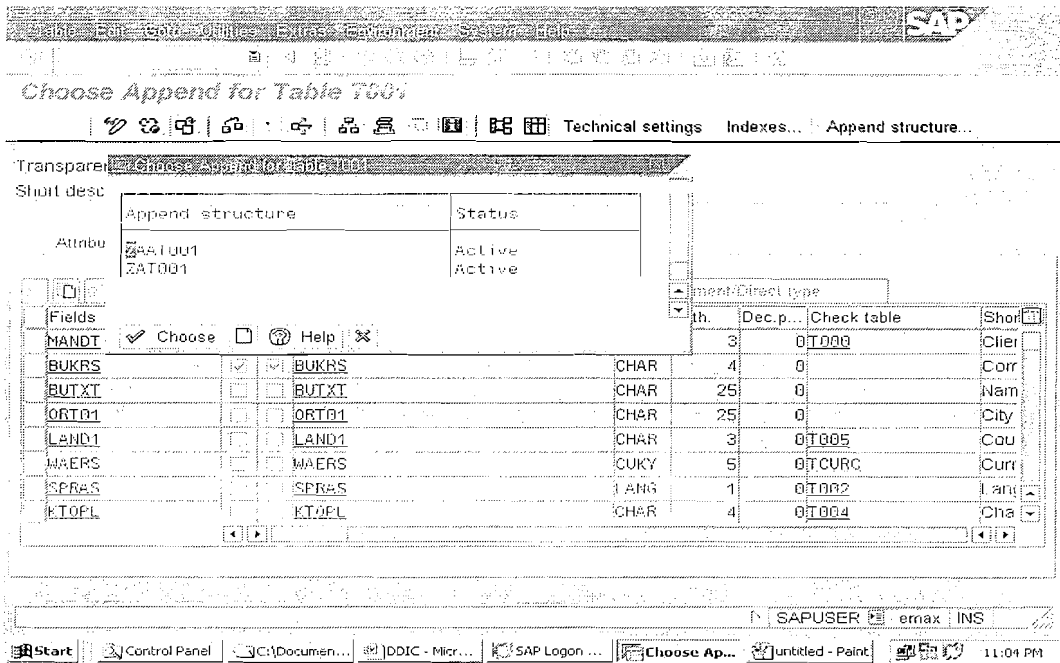
- 1) Open SE11 and 'Display' the table to which the structure is to be appended.




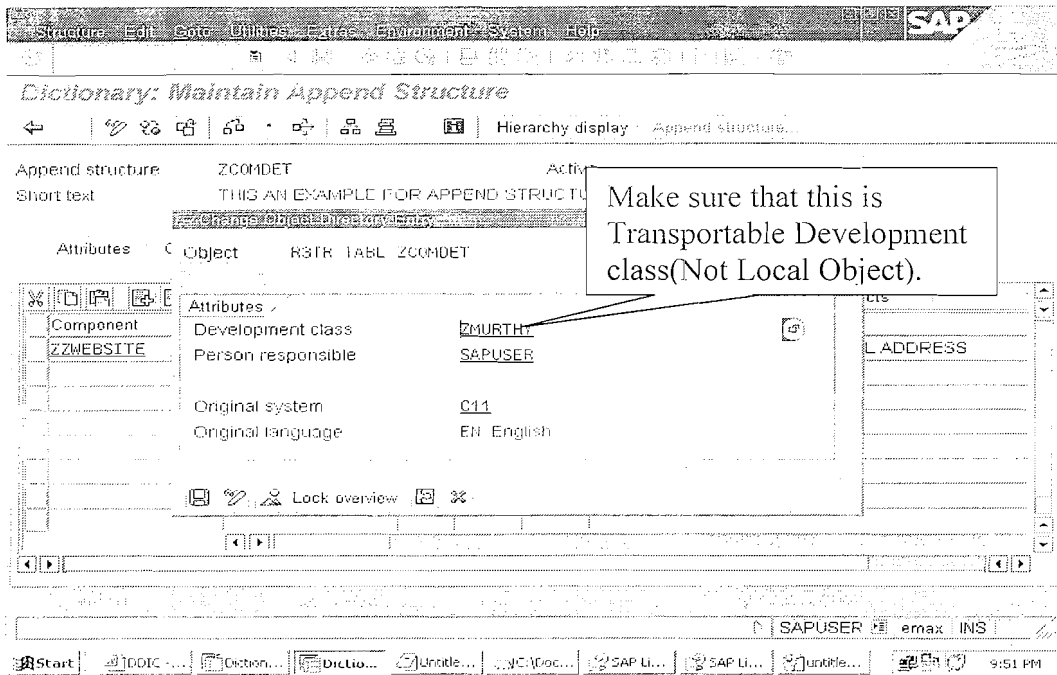
Click on 'Display'.



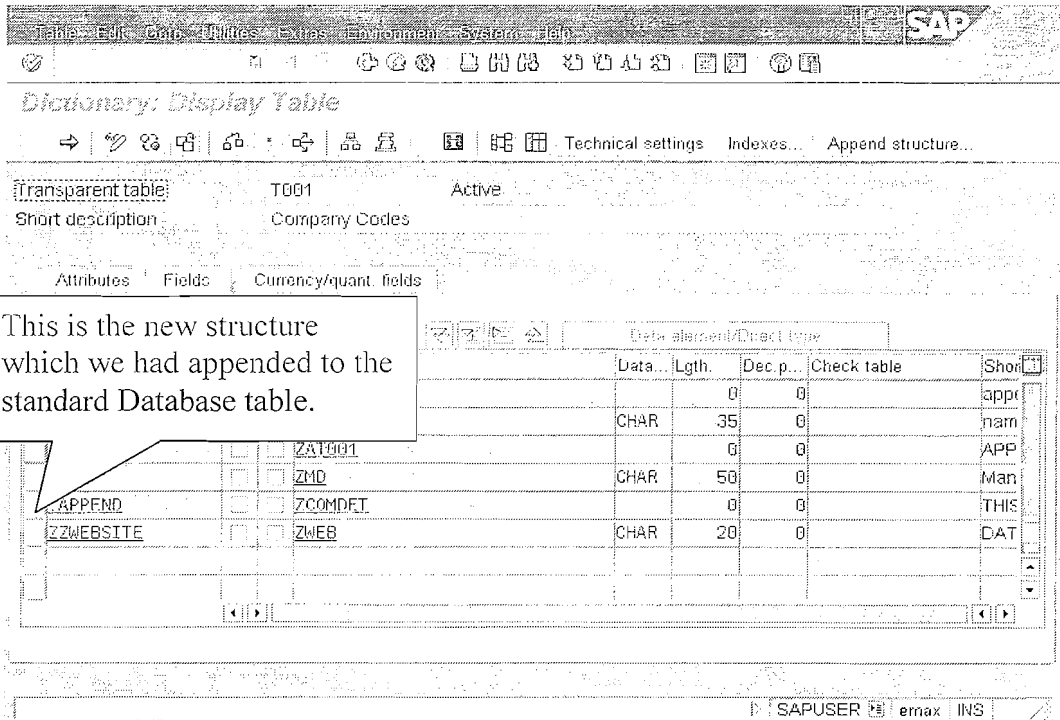
Then click on the Append structure... button on the 'Application Toolbar'. Then following screen appears.



Then click on the 'New Append Structure'  for appending structure to the standard Database table.



Save the Structure, 'Check it' and Active'. Then come back to the table screen by pressing F3 or the back button.



Working With FOREIGN KEY:-

Definition : Is a Field in one Table that is connected to another table via **Foreign Key** relationship and the purpose is to validate the data being entered in one Table (**Foreign Key** Table) with a valid set of values from another table(**Check** Table).

Let us analyze the situation where we have to create the foreign key Relationship Entries in

ZGLFA1 – Table

LIFNR	NAME1	ORT01
V00001	eMax Technologies	Hyderabad
V00002	Clarion Park	Hyderabad

Note: This is the master table for vendors i.e. these are the only vendors that we deal always.

ZGPURCHASE –Table to Maintain the Purchasing Details of the Vendor

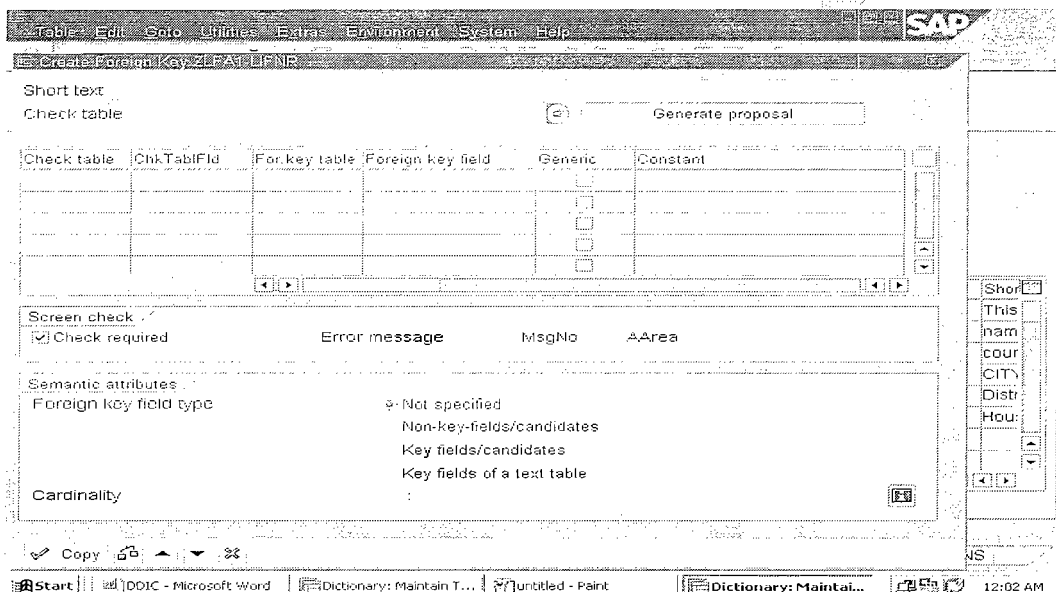
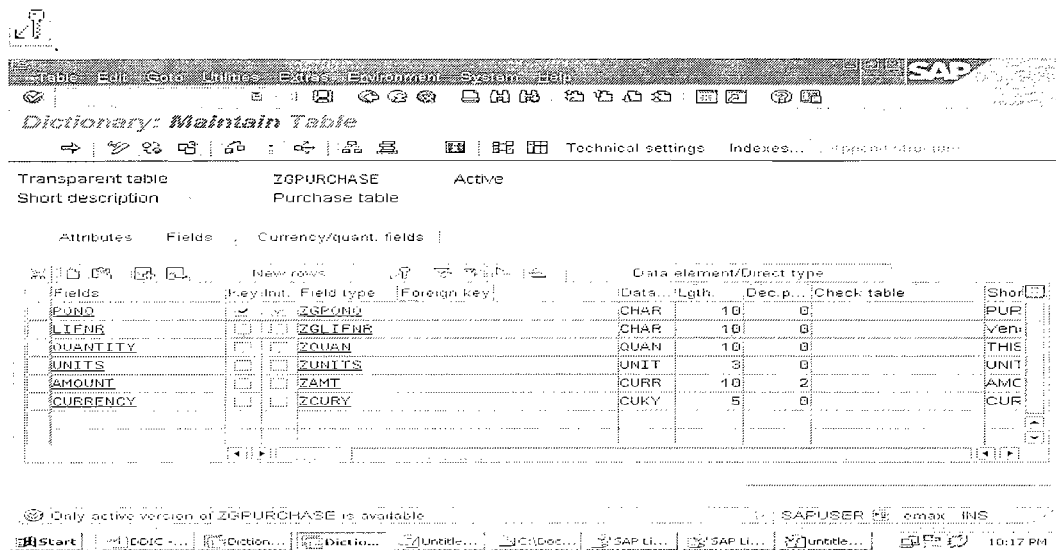
PO Number	LIFNR	AMOUNT
P00001	V00005	10000
P00002	V00007	5000

Note: The system accepts this purchase order details if we don't have the Foreign Key relationship between **ZGPURCHASE, ZGLFA1**. But in business sceneriao it should not happen to maintain the purchase orders for the vendors which are not maintained in the master table. This should be always avoided and this only possible through establishing a Foreign Key relationship between **LIFNR** of **ZGPURCHASE** and **ZGLFA1**.

Technical Requirements to create a Foreign Key:-

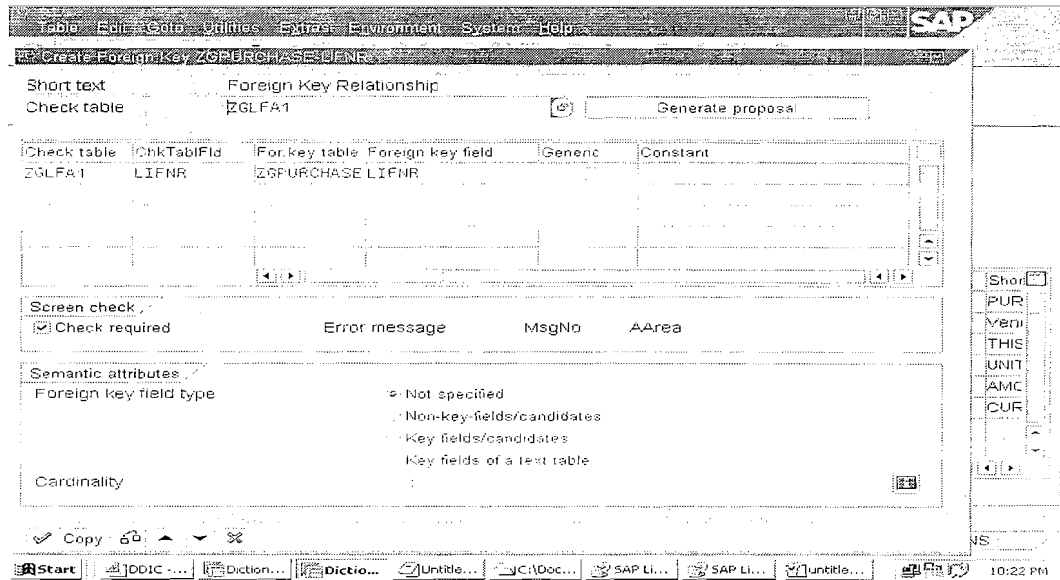
- a) The Domain names of the field in the Foreign Key table and for the field in the Check table should be same.
 - b) The field in the check table should be Primary Key.
- Steps to create Foreign Key Relationship.

Step 1. Open the Foreign Key table and select the Foreign Key field. Click on

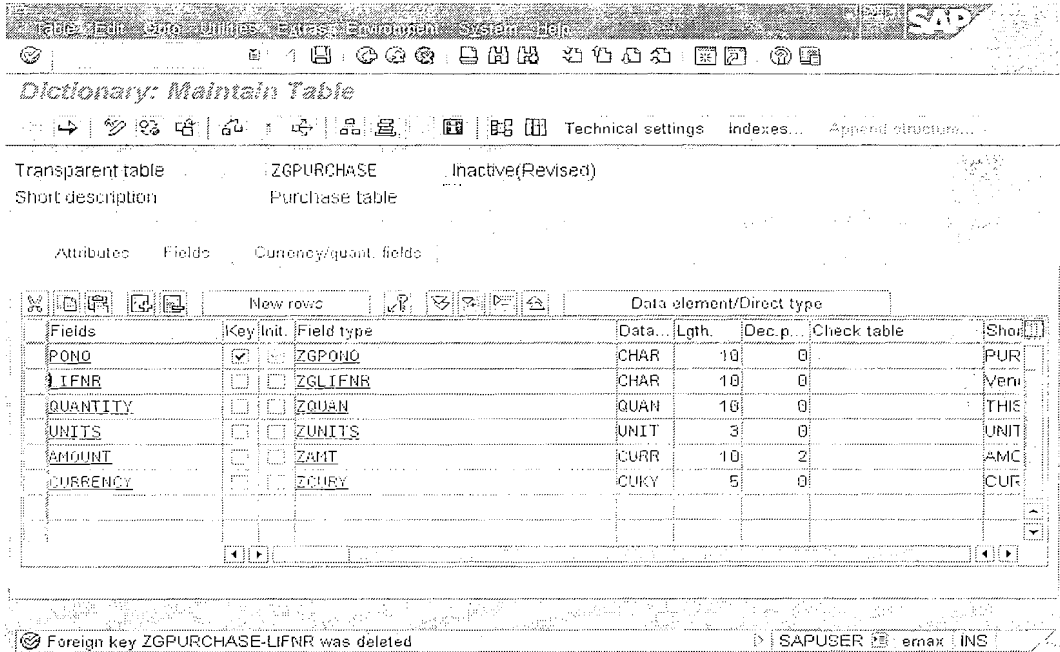


Provide the Check table name, short text and Click on ‘Generate Proposal’ button.

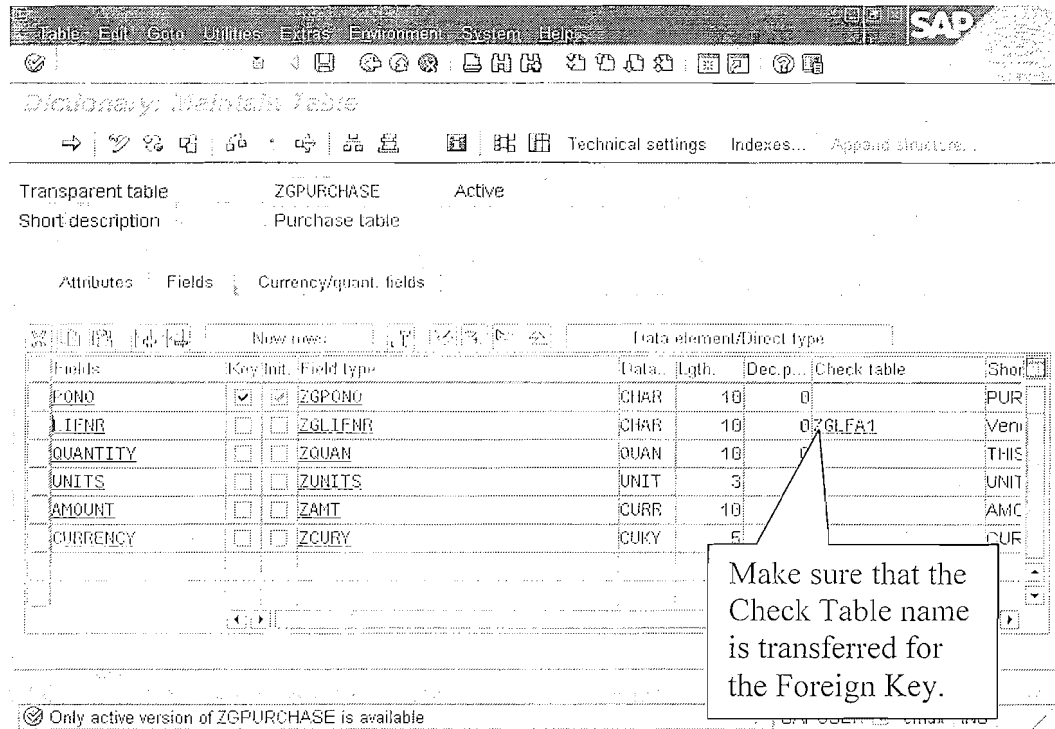
The system Generates a Proposal with the Foreign Key and all the Primary Key fields from Check table whose Domain name are same with the Foreign Key Domain.



Press 'Enter' or click on the 'Copy' button. Foreign Key relation is established between ZGLFA1 and ZGPPURCHASE



This the fig. before creating Foreign Key relation



This is the fig. after establishing Foreign Key Relation.

Triggering Foreign Key:- This will be triggered through GUI only i.e. the data will be validated for the **input provided through screens only** . It will not validate the Foreign key relation when u Insert the data into Foreign key table through Open SQL that is the reason SAP always suggest to update the Data Base Tables only through Transactions (Screens).

Working with Currency(Amount) and Quantity fields:

When we are working with international customers and Vendors if you provide only the Amount figure i.e. 1000, but is not enough as it is not clear whether it is 1000 INR,1000 USD,1000 EUR, 1000 MYR etc. So you have to always provide the currency key(INR,USD etc),when we are working with Amounts.

Similarly when we are working with quantity we have to provide Units of the quantity.

Note : When the Amount Fields are created by using Data type **CURR** and the Quantity Fields by Using **QUAN**. Then only the system expects Currency Key(**CUKY**) as reference for Amount and Units key(**UNIT**) as reference for Quantity. This is must to create a table Currency and Quantity fields

Field Name	Data Type	Reference Data type
AMOUNT	CURR	CUKY
QUANTITY	QUAN	UNIT

Note : Now we will add Quantity Field to ZGPURCHASE.

Open the table ZGPURCHASE in Change Mode and Click on New Rows to add the Quantity Field.

Table Edit Goto Utilities Extras Environment System Help

Dictionary: Maintain Table

Transparent table ZGPURCHASE Inactive(Revised)
Short description Purchase table

Attributes Fields Currency/quant. fields

New rows

Fields	Key	Init	Field type	Data	Lgth.	Dec. p...	Check table	Short
PORG	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	ZGPORG	CHAR	10	0		PUR
LIENR	<input type="checkbox"/>	<input type="checkbox"/>	ZGLIENR	CHAR	10	0	ZGLFA1	Ven
AMOUNT	<input type="checkbox"/>	<input type="checkbox"/>	ZAMT	CURR	10	2		AMC
CURRENCY	<input type="checkbox"/>	<input type="checkbox"/>	ZCURY	CUKY	5	0		CUR
QUANTITY	<input type="checkbox"/>	<input type="checkbox"/>	ZQUAN	QUAN	10	0		THS

SAPUSER emax INS

Create Data element ZQUAN and When you create the Domain , make sure that the Data Type should be **QUAN** as below.

Dictionary: Maintain Domain

Domain: ZQUAN Active
Short text: QUANTITY DOMAIN

Attributes: Definition Value range

Formatting

Data type	QUAN	Quantity field, points to a unit field with format UNIT
No. characters	10	
Decimal places		

Output characteristics

Output length	13
Convers. routine	
<input type="checkbox"/> Sign	
<input type="checkbox"/> Unrestricted	

SAPUSER emax INS

And also create another field(Reference field for Quantity) with Data type UNIT. When you create reference field domain for Quantity, make sure that the Data should be **UNIT** as below.

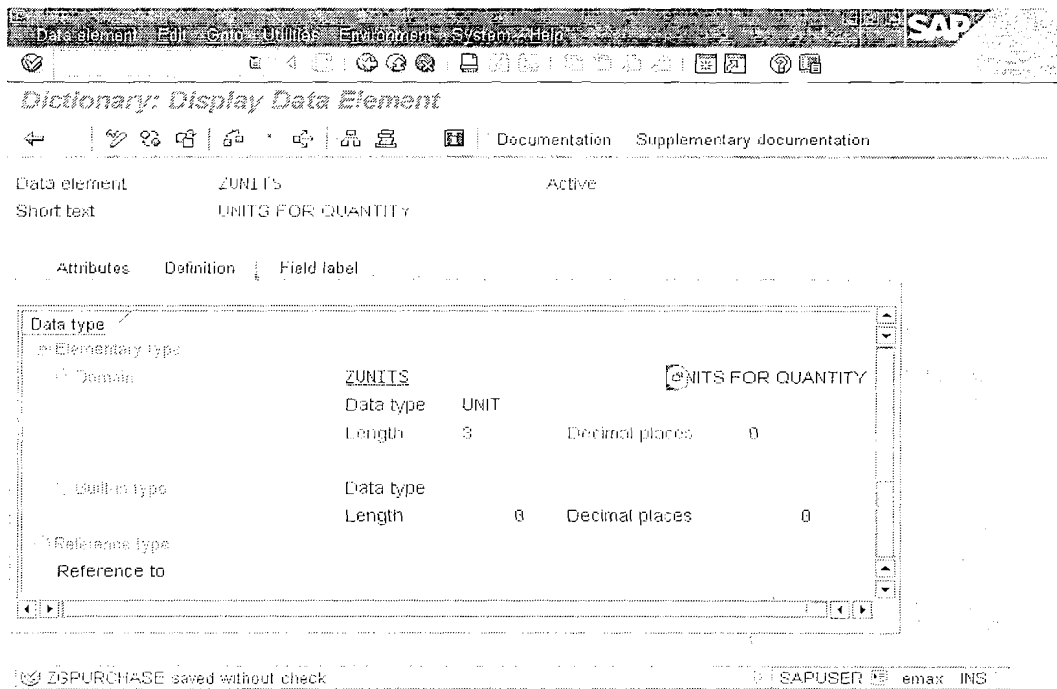
Dictionary: Maintain Table

Transparent table: ZGPURCHASE Inactive
Short description: Purchase table

Attributes: Fields Currency/quant. fields

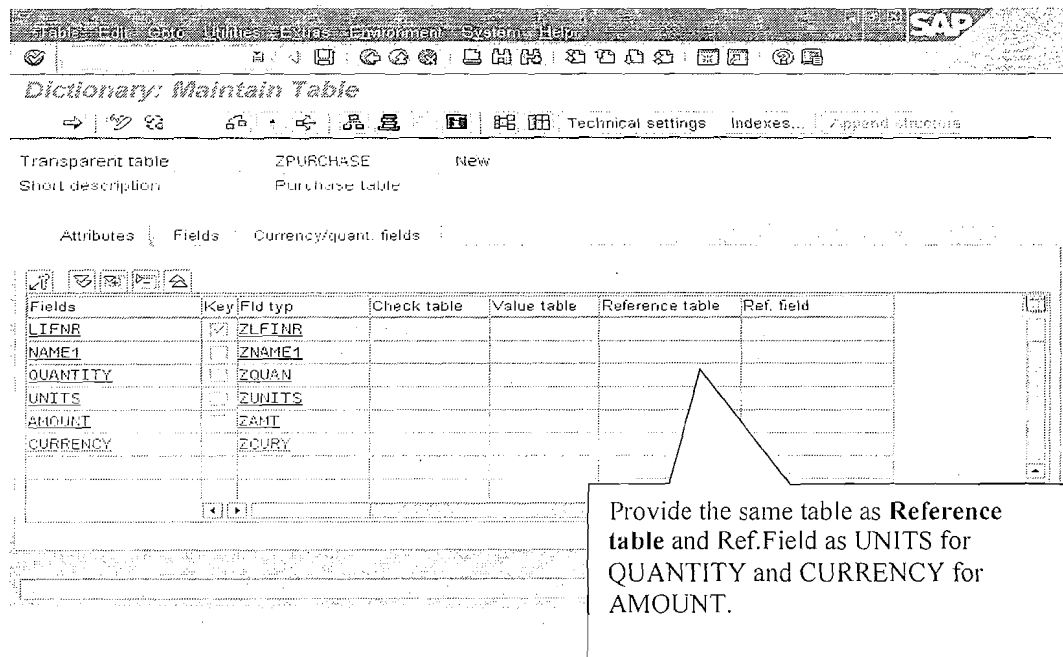
Fields	Key	Init.	Field type	Data..	Lgth.	Dec. p...	Check table	Short
PONO	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	ZGPONO	CHAR	10	0		PUR
LIFNR	<input type="checkbox"/>	<input type="checkbox"/>	ZGLIFNR	CHAR	10		ZGLFA1	Veri
AMOUNT	<input type="checkbox"/>	<input type="checkbox"/>	ZAMT	CURR	10	2		AMC
CURR BLY	<input type="checkbox"/>	<input type="checkbox"/>	ZCURY	CURY	5	0		CUR
QUANTITY	<input type="checkbox"/>	<input type="checkbox"/>	ZQUAN	QUAN	10	0		THIE
UNITS	<input type="checkbox"/>	<input type="checkbox"/>	ZUNITS	UNIT	3	0		UNIT

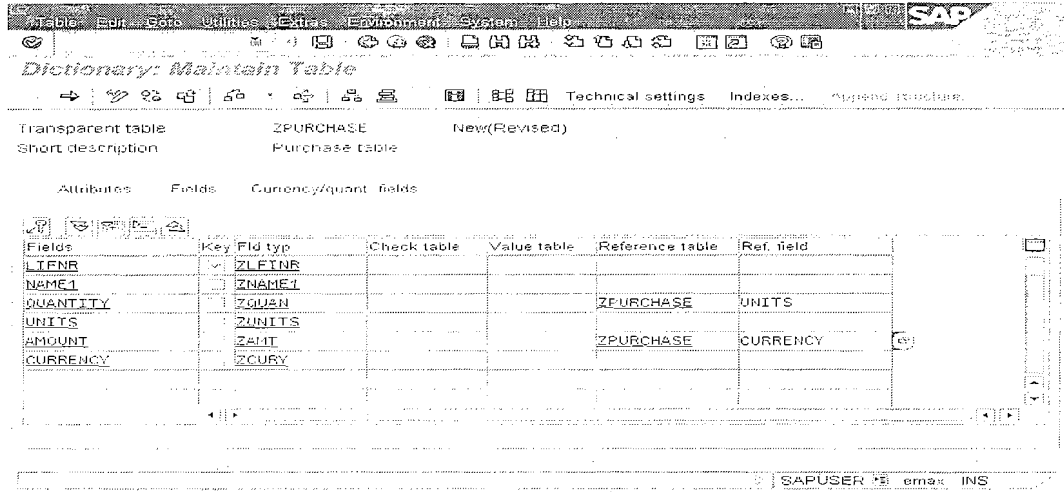
SAPUSER emax INS



Note : Repeat the Same for AMOUNT field it is 'CURR' and for Currency field it is 'CUKY'.

Now click on the tab **Currency/quant. fields**





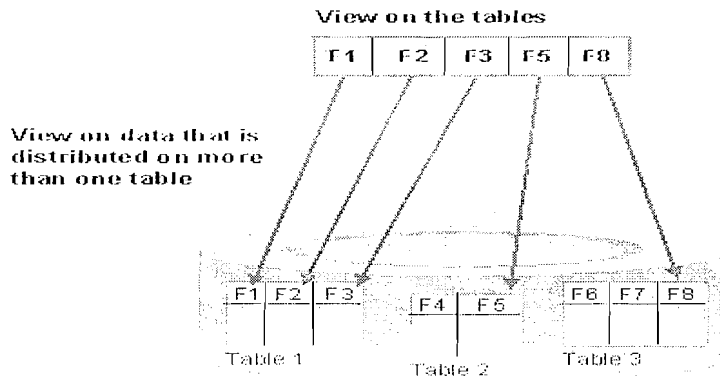
This is how 'Currency And Quantity Fields' are related to the units of the particular fields.

VIEWS:-

Views

Views are logical views on more than one table. The structure of the view is defined in the ABAP Dictionary. A view on the database can then be created from this structure.

- Data about an application object is often distributed on several tables.
- By defining a view, you can define an application-dependent view that combines this data.
- The structure of such a view is defined by specifying the tables and fields used in the view.
- Fields that are not required can be hidden, thereby minimizing interfaces.
- A view can be used in ABAP programs for data selection.



The data of a view is derived from one or more tables, but not stored physically. The simplest form of deriving data is to mask out one or more fields from a base table (projection) or to include only certain entries of a base table in the view (selection).

Four different view types are supported. These differ in the way in which the view is implemented and in the methods permitted for accessing the view data.

- **Database views** are implemented with an equivalent view on the database.

Data about an application object is often distributed on several database tables. A database view provides an application-specific view on such distributed data.

Database views are defined in the ABAP Dictionary. A database view is automatically created in the underlying database when it is activated.

- **Projection views** are used to hide fields of a table (only projection).

Projection views are used to hide fields of a table. This can minimize interfaces; for example when you access the database, you only read and write the field contents actually needed.

A projection view contains exactly one table. You cannot define selection conditions for projection views.

- **Help views** can be used as selection method in search helps.

You have to create a help view if a view with outer join is needed as selection method of a

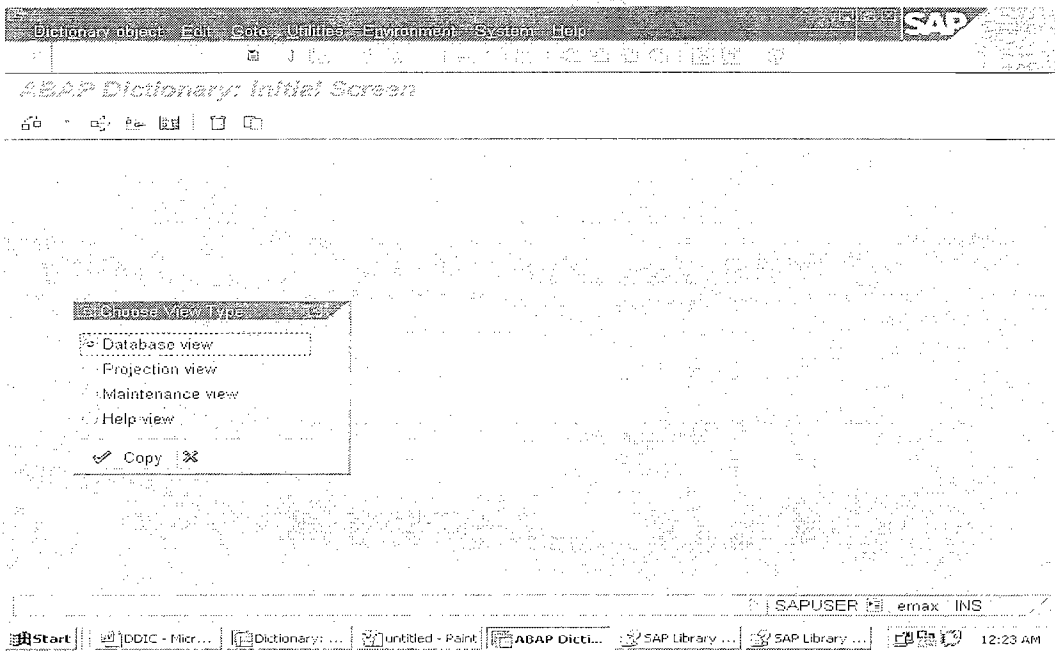
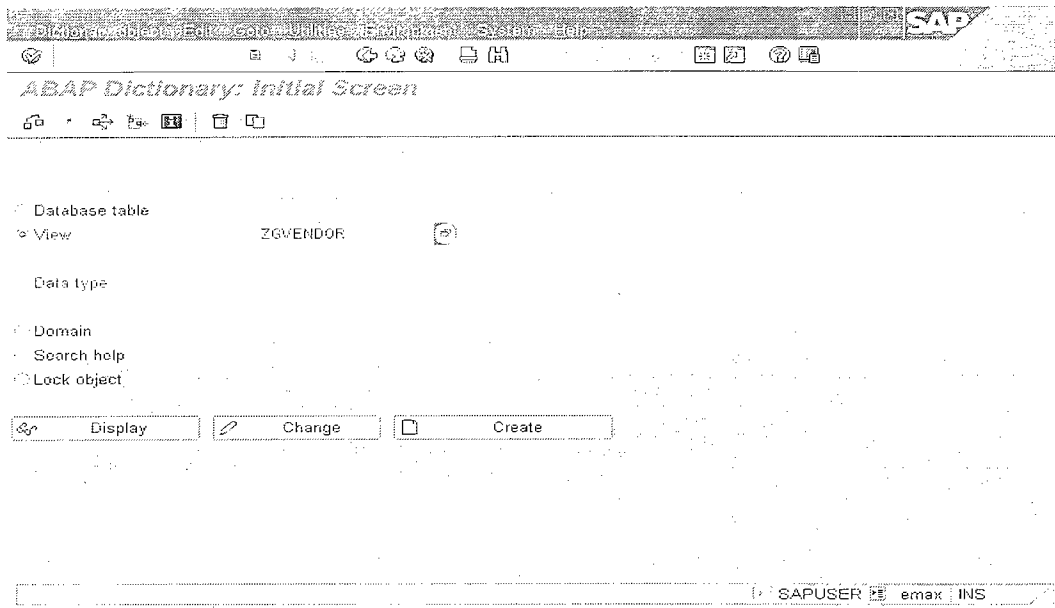
search help.

- **Maintenance views** permit you to maintain the data distributed on several tables for one application object at one time.

A maintenance view permits you to maintain the data of an application object together. The data is automatically distributed in the underlying database tables. The maintenance status determines which accesses to the data of the underlying tables are possible with the maintenance view.

DATABASE VIEW (Most of the times we work with Data base View only):-

Open SE11, select the option 'View' on the screen . Enter the name of the 'View' to be created and click on 'Create'.

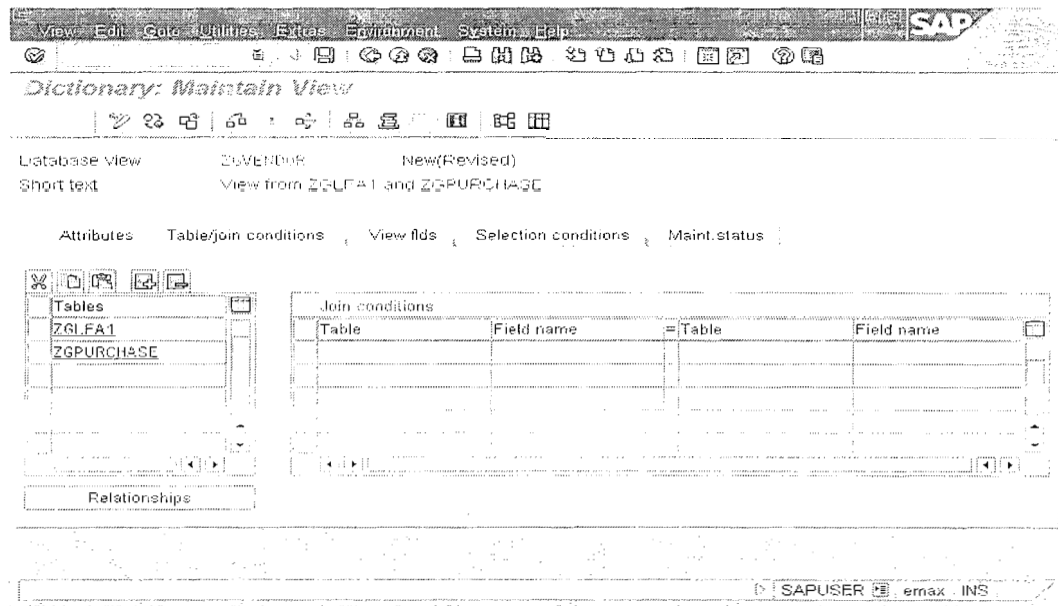


Click on the **Database View** and say ‘Enter’ or click on the ‘Copy’ button.

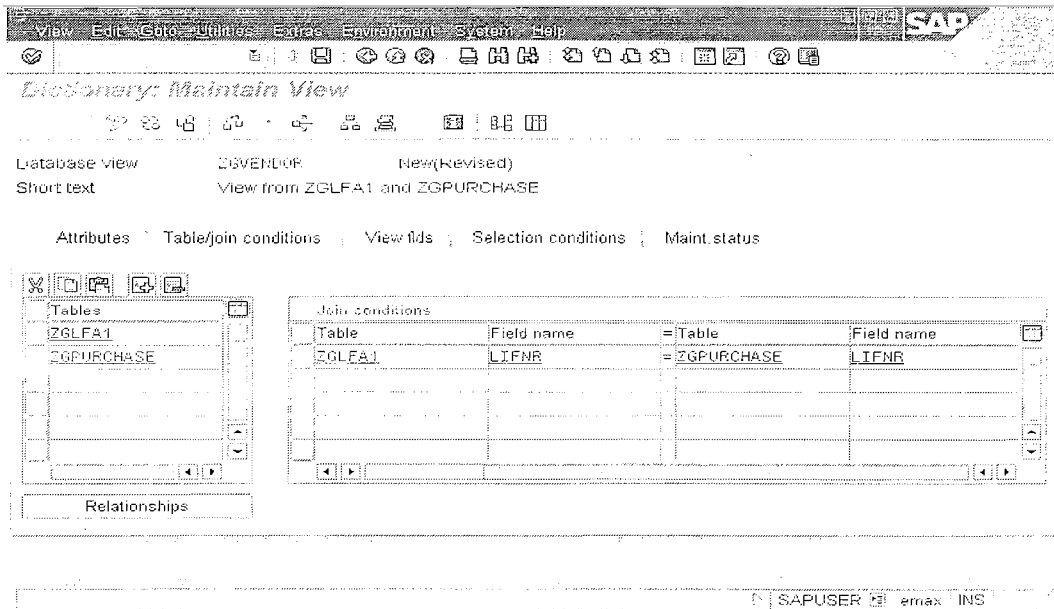
Enter the Short text and table names in the ‘**Tables**’ for which view has to be created.

Select the tables from where we derive the **View** and Click on the



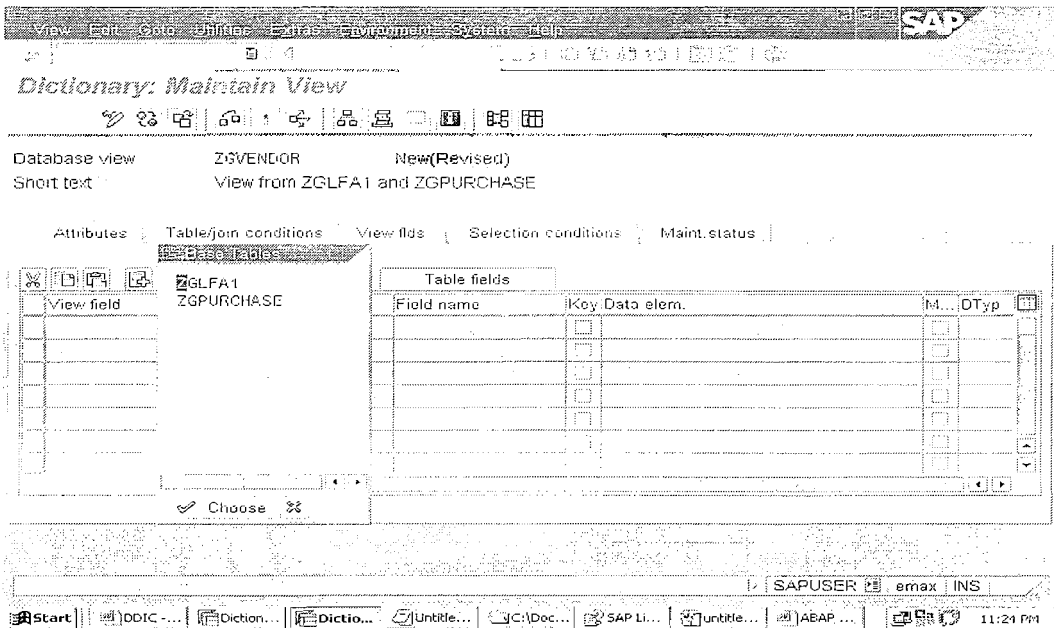


Click on the 'Copy' button or press 'Enter'. A list of fields which are having foreign key related fields are displayed.



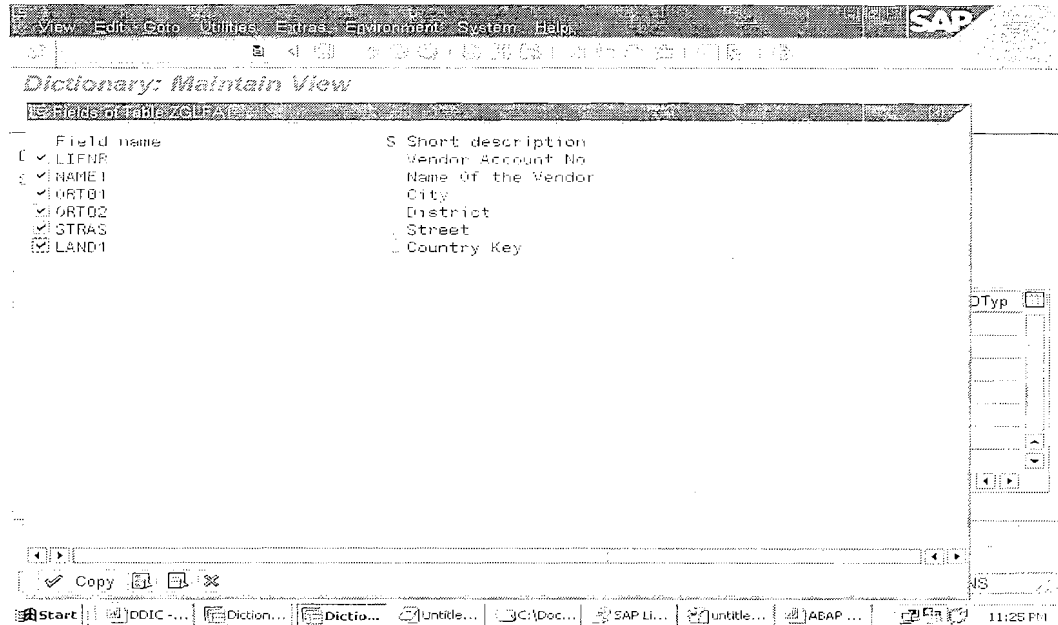
Then click on the **view fids** tab to select the list of the fields from the tables that are to be displayed along with Primary and Foreign Key Fields.

Note: Here either we manually enter the field names and the table names that are to be displayed or we can click **Table fields** where the system proposes the tables. From there we can select the fields from each table.

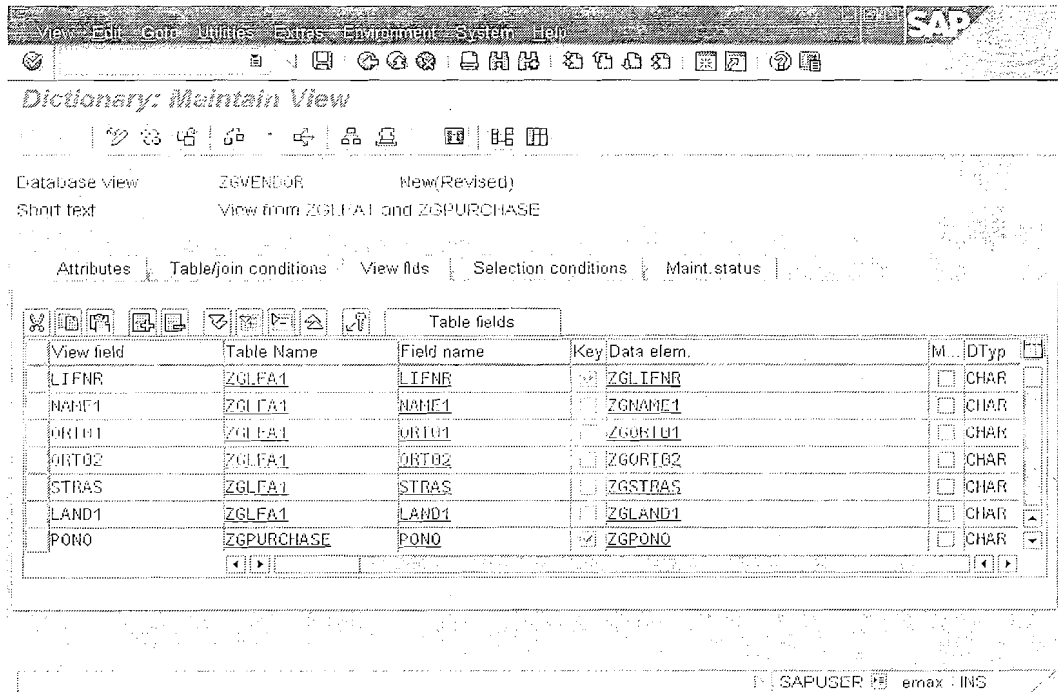


Click on the Table, and **'Choose'** button. List of fields of the table are displayed. Select the fields and say **'Copy'** or press **'Enter'**. And again Click the button

'Table Fields' for selecting the fields from the second table. Select the fields from the second table say 'Copy' same as the above.



Repeat the same for other tables also.



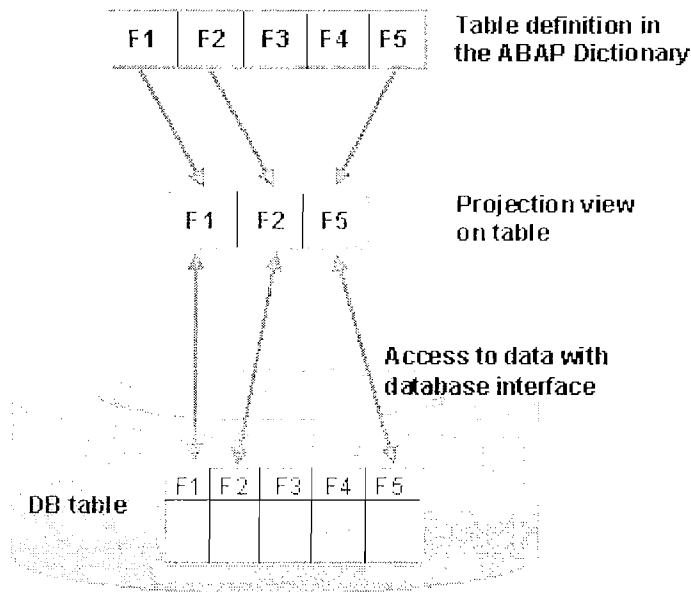
Save, Check, Activate.

Projection Views

Projection views are used to hide fields of a table. This can minimize interfaces; for example when you access the database, you only read and write the field contents actually needed.

A projection view contains exactly one table. You cannot define selection conditions for projection views.

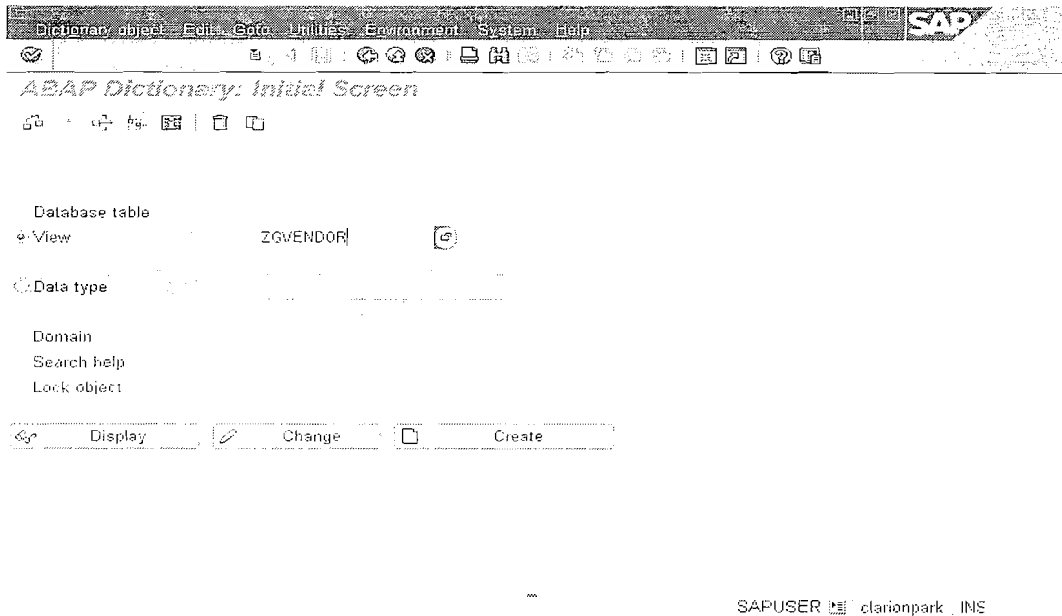
There is no corresponding object in the database for a projection view. The R/3 System maps the access to a projection view to the corresponding access to its base table. You can also access pooled tables and cluster tables with a projection view.



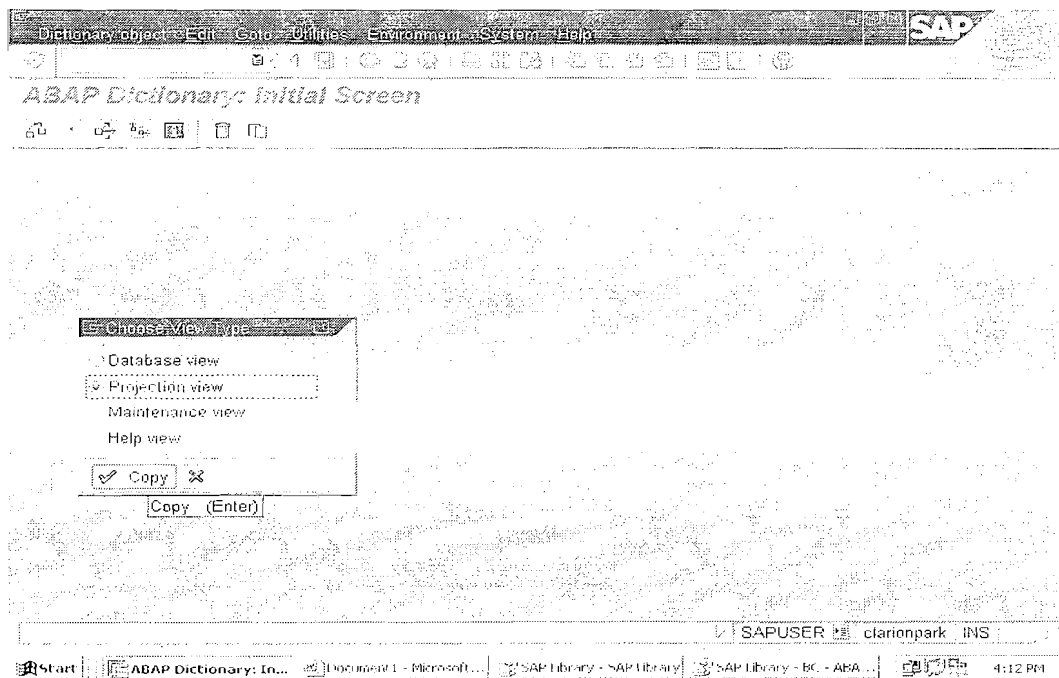
The maintenance status of the view controls how the data of the table can be accessed with the projection view.

Steps To create Projection View:-

- 1) Execute SE11 and select the radio button View and provide the View Name.

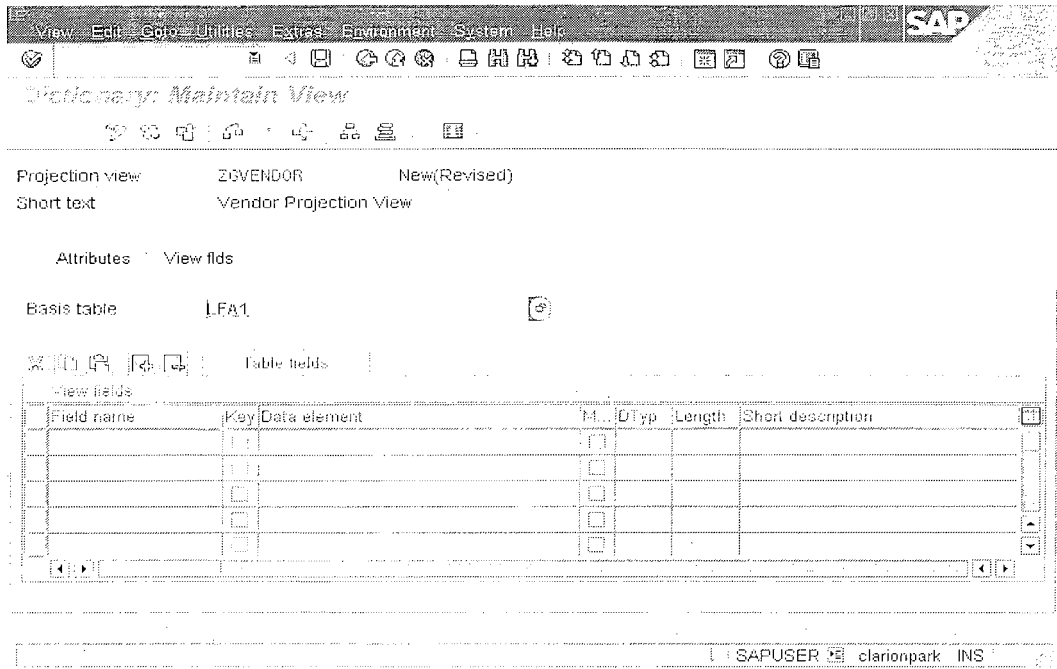


Click on Create and make sure that you select Projection View Radio Button ,‘Enter’.

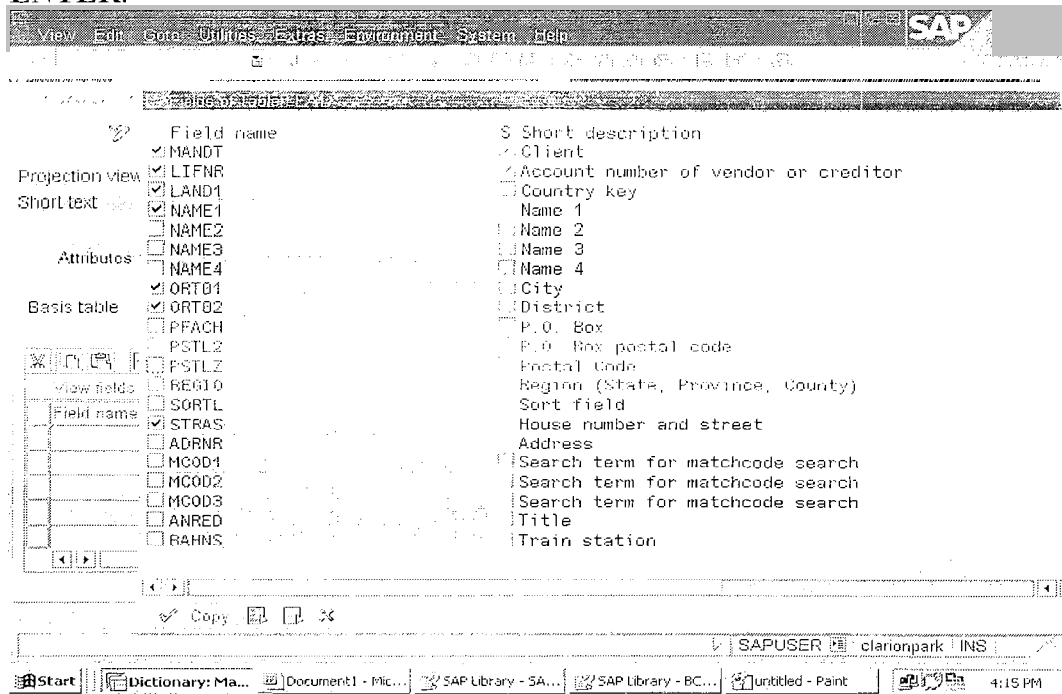


Enter an explanatory short text in the field *Short text*

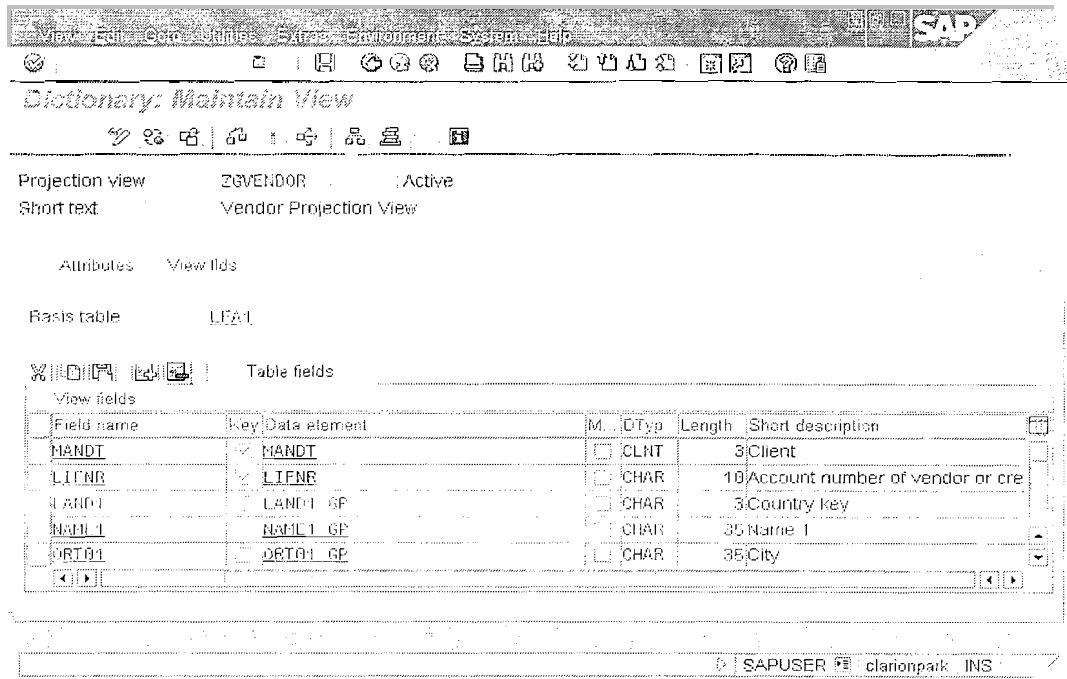
Enter a table name in the field *Base table*.(A projection view always contains exactly one table.)



Click on the **Table fields** and select the required fields to be displayed, **ENTER**.

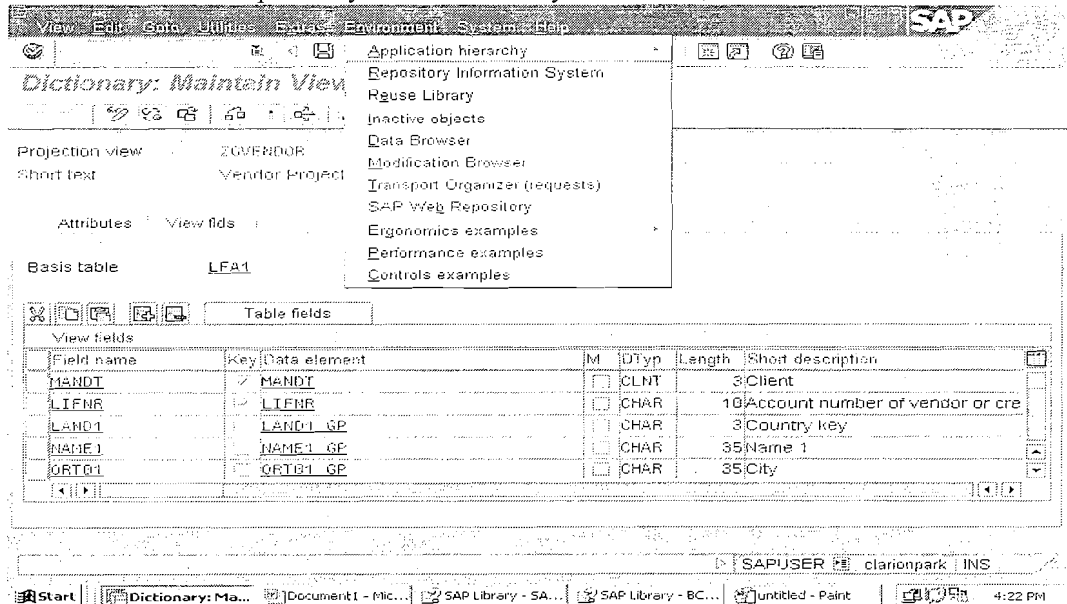


Save it , Check and Activate.



Steps to display the contents from the View.

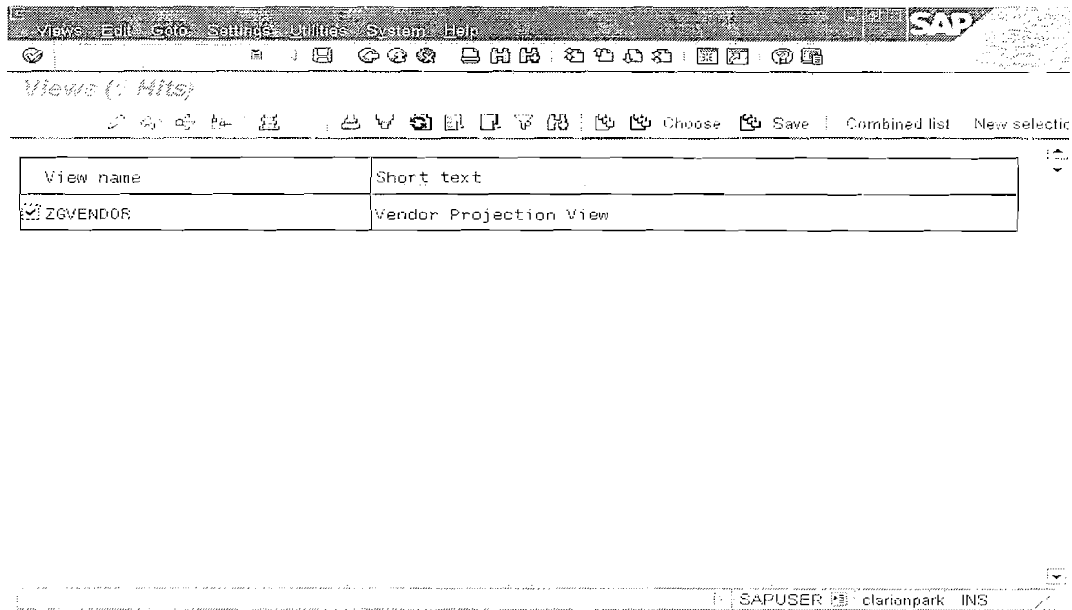
Environment -> Repository Information System.



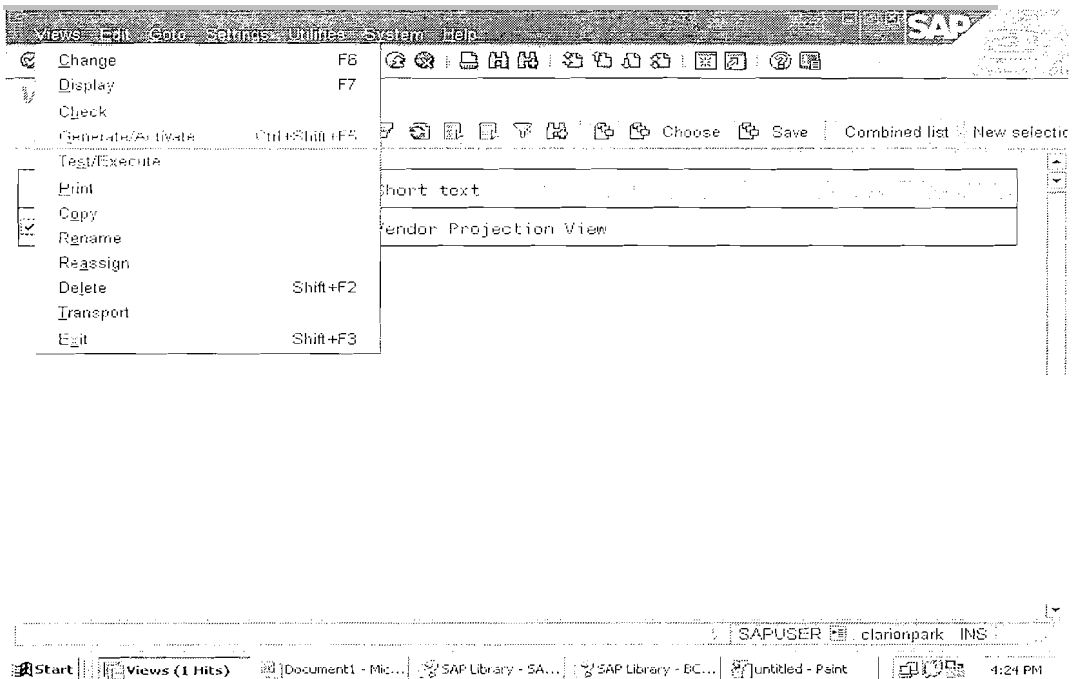
Click on ABAP Dictionary -> Basic Obj's -> Views.

The screenshot shows the SAP R/3 Repository Information System interface. The top menu bar includes 'Object', 'Edit', 'Tools', 'Settings', 'Utilities', 'Environment', 'System', and 'Help'. The main window displays a tree view of the R/3 Repository Information System, with 'ABAP Dictionary' expanded to show 'Views'. A dialog box titled 'Enter the View Name' is open, showing a search for 'egvvendor'. The dialog has two sections: 'Standard selections' and 'Settings'. The 'Standard selections' section has fields for 'View name' (containing 'egvvendor'), 'Short description', and 'Development class'. The 'Settings' section has a field for 'Maximum no. of hits' set to '200'. The bottom status bar shows 'SAPUSER', 'clarionpark', and 'INS'.

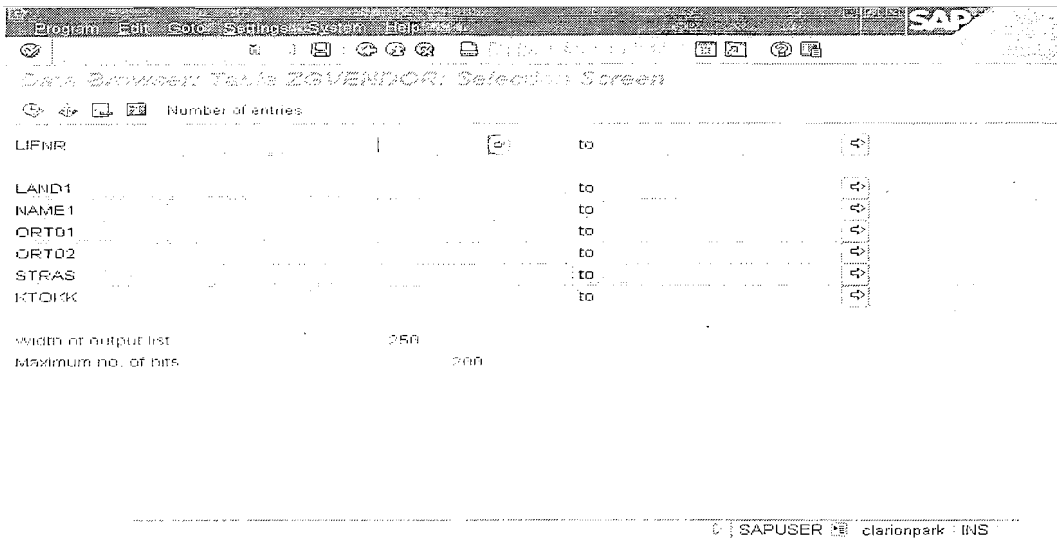
Select the View name from which Data is to be displayed.



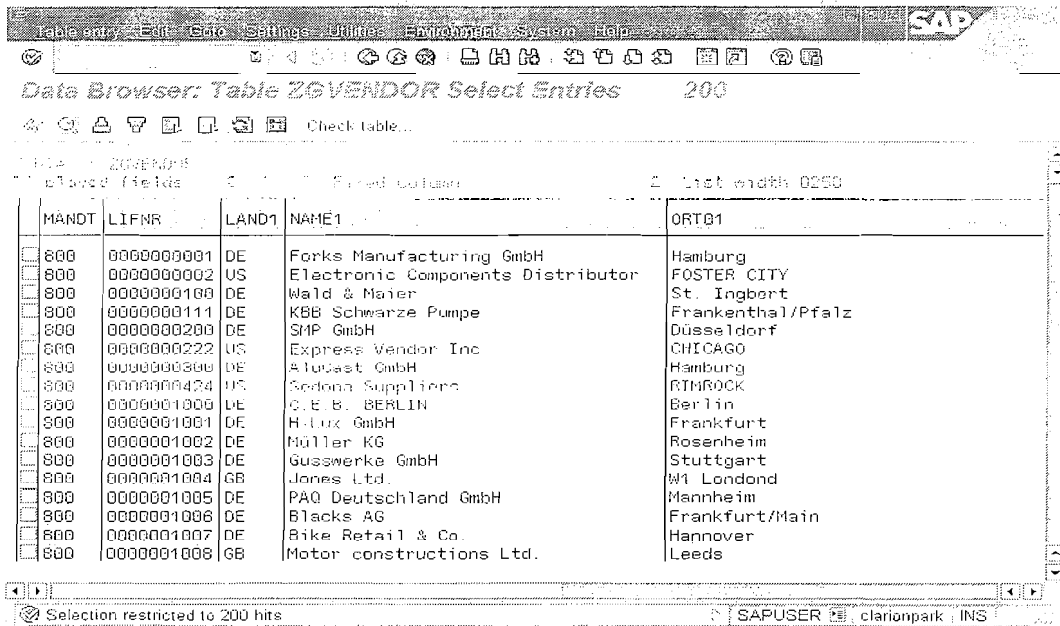
Menu Path View -> Test / Execute.



Execute.



Final Output.



Lock Objects

The R/3 System synchronizes simultaneous access of several users to the same data records with a lock mechanism. When interactive transactions are programmed, locks are set and released by calling function modules. These function modules are automatically generated from the definition of lock objects in the ABAP Dictionary.

Structure of a Lock Object

The tables in which data records should be locked with a lock request are defined in a lock object together with their key fields. When tables are selected, one table (the primary table) is first selected. Further tables (secondary tables) can also be added using foreign key relationships.

Lock Arguments

The lock argument of a table in the lock object consists of the key fields of the table.

The lock argument fields of a lock object are used as input parameters in the function modules for setting and removing locks generated from the lock object definition. When these function modules are called, the table rows to be locked or unlocked are specified by defining certain values in these fields. These values can also be generic. The lock argument fields therefore define which subset of the table rows should be locked.

A lock mode can be assigned for each table in the lock object. This mode defines how other users can access a locked record of the table.

Lock Mode

Access by more than one user can be synchronized in the following

- **Exclusive lock:** The locked data can only be displayed or edited by a single user. A request for another exclusive lock or for a shared lock is rejected.
- **Shared lock:** More than one user can access the locked data at the same time in display mode. A request for another shared lock is accepted, even if it comes from another user. An exclusive lock is rejected.
- **Exclusive but not cumulative:** Exclusive locks can be requested several times from the same transaction and are processed successively. In contrast, exclusive but not cumulative locks can be called only once from the same transaction. All other lock requests are rejected.

Parameters of the Function Modules

Field Names of the Lock Object

The keys to be locked must be passed here.

A further parameter $X_{<field>}$ that defines the lock behavior when the initial value is passed exists for every lock field $<field>$. If the initial value is assigned to $<field>$ and $X_{<field>}$, then a generic lock is initialized with respect to $<field>$. If $<field>$ is assigned the initial value and $X_{<field>}$ is defined as X , the lock is set with exactly the initial value of $<field>$.

Parameters for Passing Locks to the Update Program

A lock is generally removed at the end of the transaction or when the corresponding DEQUEUE function module is called. However, this is not the case if the transaction has called update routines. In this case a parameter must check that the lock has been removed.

Parameter `_SCOPE` controls how the lock or lock release is passed to the update program (see The Owner Concept for Locks). You have the following options:

- `_SCOPE = 1`: Locks and lock releases are not passed to the update program. The lock is removed when the transaction is ended.
- `_SCOPE = 2`: The lock or lock release is passed to the update program. The update program is responsible for removing the lock. The interactive program with which the lock was requested no longer has an influence on the lock behavior. This is the standard setting for the ENQUEUE function module.
- `_SCOPE = 3`: The lock or lock release is also passed to the update program. The lock must be removed in both the interactive program and in the update program. This is the standard setting for the DEQUEUE function module.

Parameters for Lock Mode

A parameter `MODE_<TAB>` exists for each base table `TAB` of the lock object. The lock mode for this base table can be set dynamically with this parameter. Valid values for this parameter are *S* (shared), *E* (exclusive) and *X* (exclusive but not cumulative).

The lock mode specified when the lock object for the table is created is the default value for this parameter. This default value can however be overridden as required when the function module is called.

If a lock set with a lock mode is to be removed by calling the DEQUEUE function module, this call must have the same value for the parameter `MODE_<TAB>`.

Controlling Lock Transmission

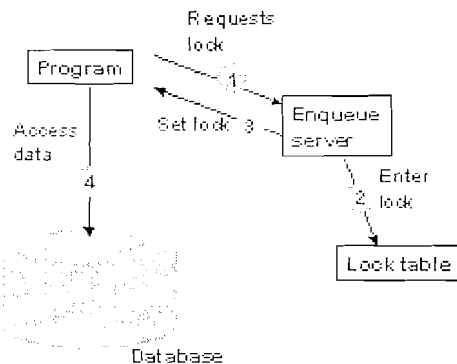
Parameter `COLLECT` controls whether the lock request or lock release should be performed directly or whether it should first be written to the local lock container. This parameter can have the following values:

- **Initial value:** The lock request or lock release is sent directly to the lock server.
- **X** : The lock request or lock release is placed in the local lock container. The lock requests and lock releases collected in this lock container can then be sent to the lock server at a later time as a group by calling the function module `FLUSH_ENQUEUE`.
- **Lock Mechanism**

You can synchronize access by several programs to the same data with a logical lock mechanism. This lock mechanism fulfills two main functions:

- A program can tell other programs which data records it is just reading or changing.
- A program can prevent itself from reading data that is just being changed by another program.

The data records of a table to be locked are defined by a logical condition. When a lock is set, this logical condition is entered in a lock table. This entry is retained until it is removed by the program or the program comes to an end. All the locks set by a program are thus removed at the end of the program.

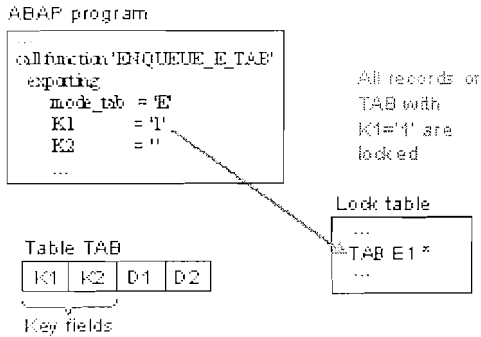


When accessing data records, the records just being edited by other programs can be identified by the entry in the lock table. Such an entry for the lock must define a number of fully specified key fields, that is either a value is passed for the key field or this field is locked generically.

To set locks, a lock object must be defined in the ABAP Dictionary. When this lock object is activated, two function modules are generated with the names *ENQUEUE_<lockobjectname>* and *DEQUEUE_<lockobjectname>*.

If data records are to be locked, you must call function module *ENQUEUE_<lockobjectname>*. The values of the key fields that specify the records to be locked are passed for all the tables contained in the lock object when the function module is called. There is a generic lock if a value is not passed for all the key fields. The function module writes the appropriate lock entry. If another program also requests a lock, it will be accepted or rejected depending on the lock mode. The program can then react to this situation.

Locked data records can be unlocked by calling function module *DEQUEUE_<lockobjectname>*. The key values and the lock mode used to set the lock must be passed to the function module.



This lock procedure requires that all programs involved cooperate. Inconsistencies can occur if a program reads or changes data without having previously locked it. When a lock is set, the data records are only protected against changes by another program if this program also requests a lock before accessing the data.

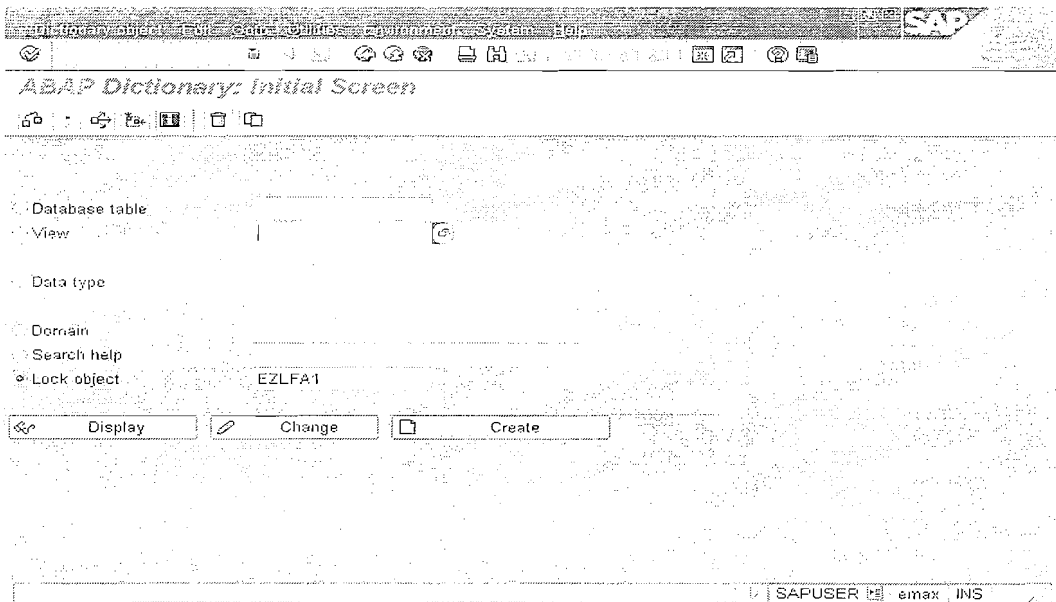
Instead of writing lock requests or lock releases directly in the lock table, it is also possible to collect them first in a local lock container. The collected locks can be sent at a later time as a group. A parameter of the relevant function module controls whether a lock request or lock release is sent directly.

You can find further information about the lock concept and how lock management works in the documentation *The R/3 Lock Concept*.

Creating Lock Objects

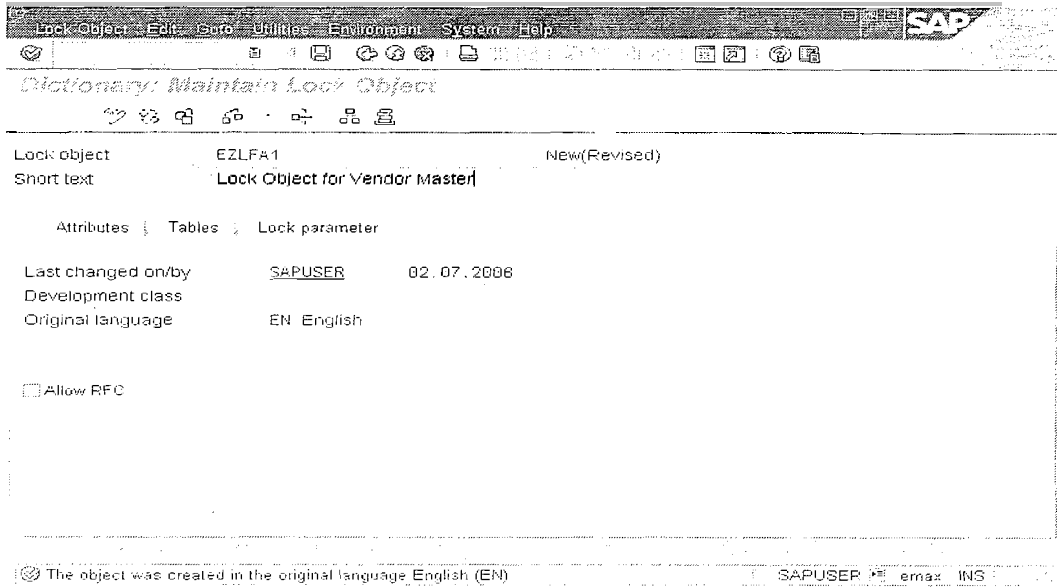
1. Select object type *Lock object* in the initial screen of the ABAP Dictionary, enter an object name and choose *Create*. The name of a lock object should begin with an E (Enqueue).

The maintenance screen for lock objects is displayed



2 Enter an explanatory short text in the field *Short text*.

You can then use the short text to find the lock object at a later time, for example with the R/3 Repository Information System.

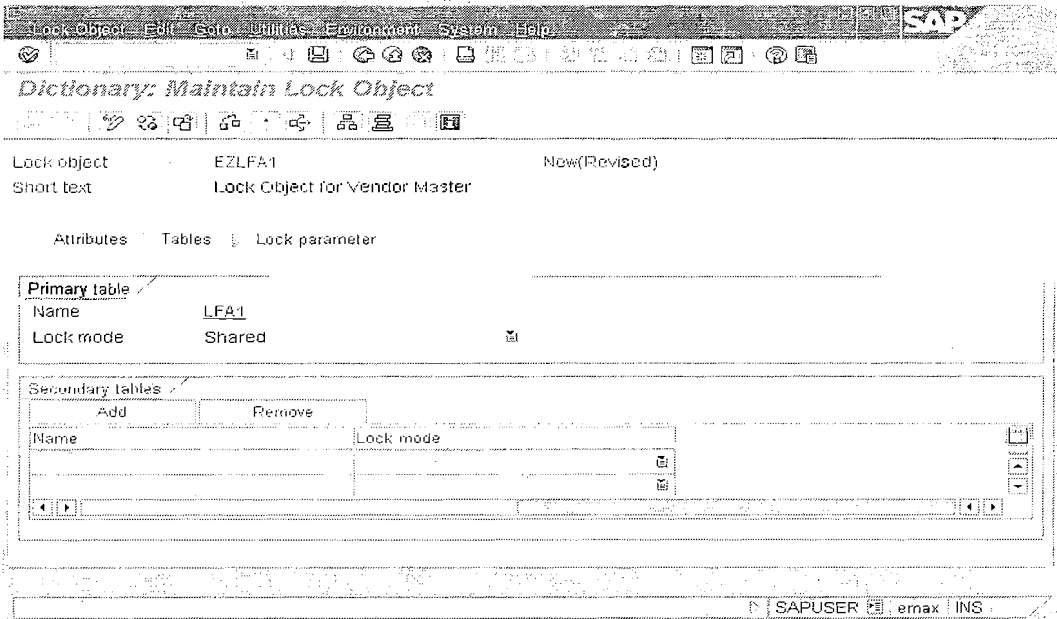


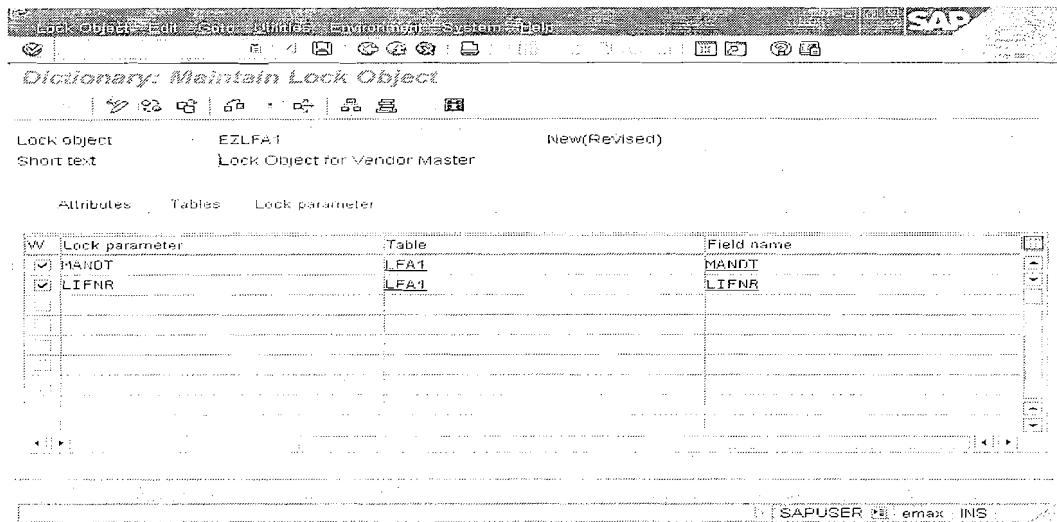
3. Enter the name of the primary table of the lock object.

All other tables in the lock object must be linked with the primary table using foreign keys. There are also some restrictions on the valid foreign key relationships.

4. Select the lock mode of the primary table in the field below it.

The lock mode is used as the default value for the corresponding parameters of the function modules generated from the lock object.





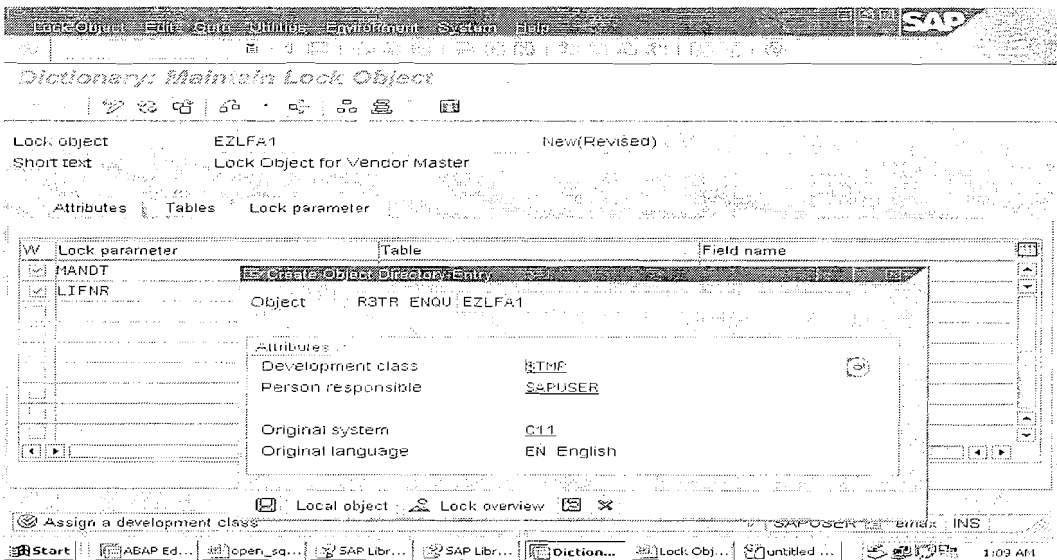
5. Choose *Add* if you want to lock records in more than one table with the lock object.

A list of all the tables linked with the primary table using valid foreign keys is displayed. Select the appropriate table. The lock mode of the primary table is copied as lock mode. You can change this setting as required, for example you can assign the lock mode separately for each table.

Similarly, you can add a table linked with the secondary table just added with foreign keys. To do this, place the cursor on the name of the secondary table and choose *Add*.

6. Save your entries.

A dialog box appears in which you have to assign the lock object a development class.



Ctrl + S , Ctrl+F2 , Ctrl + F3.

Result

You can find information about the activation flow in the activation log, which you can display with *Utilities* □ *Activation log*. If errors occurred during activation, the activation log is displayed immediately.

Check for the Function Modules Generated.

Goto->Lock Modules

Function group Name of function module	Function group short text Short text for function module
/1BC0WBEN/TEN0000 DEQUEUE_EZLFA1	Release lock on object EZLFA1
ENQUEUE_EZLFA1	Request lock for object EZLFA1

Calling Lock Objects :

Sample Program:

REPORT ZDEMO_CHECK_LOCK_OBJECT .

```

*****
* PROGRAM       : ZDEMO CHECK LOCK_OBJECT
* AUTHOR        : GANAPATI.ADIMULAM
* START DATE    : 30/06/2006
* PURPOSE       : IS TO WORK WITH ALL OPEN SQL
                  OPERATIONS
* COPIED FROM   : NA
*****
* MODIFICATION LOG:
* CHANGE REQUEST : C11EMAX4756
* -----
*MOD-001       : DETAILS OF MODIFICATION 001
    
```

```
*MOD-002      :      DETAILS OF MODIFICATION 002
*SUPPLIERS    :      EMAX TECHNOLOGIES
*****
```

```
CALL FUNCTION 'ENQUEUE_EZLFA1'
```

```
EXPORTING
```

```
MODE_LFA1      = 'E'
MANDT          = SY-MANDT
LIFNR         = '0000000001'
* X_LIFNR      = ''
* _SCOPE       = '2'
* _WAIT        = ''
* _COLLECT     = ''
```

```
* EXCEPTIONS
```

```
* FOREIGN_LOCK = 1
* SYSTEM_FAILURE = 2
* OTHERS       = 3
```

```
IF SY-SUBRC <> 0.
```

```
* MESSAGE ID SY-MSGID TYPE SY-MSGTY NUMBER SY-MSGNO
* WITH SY-MSGV1 SY-MSGV2 SY-MSGV3 SY-MSGV4.
ENDIF.
```

```
UPDATE LFA1 SET ORT01 = 'HYDERABAD'
           ORT02 = 'RANGA REDDDY'
           WHERE LIFNR = '0000000001'.
```

```
IF SY-SUBRC = 0.
```

```
WRITE : / 'RECORD IS SUCCESSFULLY UPDATED'.
```

```
*UN LOCK THE SAME ONCE THE UPDATION IS FINISHED
CALL FUNCTION 'DEQUEUE_EZLFA1'
```

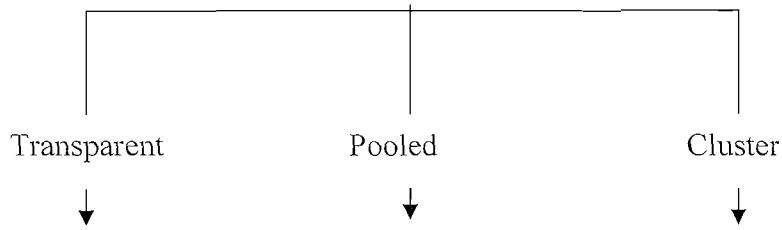
```
EXPORTING
```

```
MODE_LFA1      = 'E'
MANDT          = SY-MANDT
LIFNR         = '0000000001'
* X_LIFNR      = ''
* _SCOPE       = '3'
* _SYNCHRON    = ''
* _COLLECT     = ''
```

```
ELSE.
```

```
WRITE : / 'RECORD UPDATION IS NOT SUCCESSFUL'.
ENDIF.
```

Types of Data Dictionary Tables:-



Transparent Table:-

It is used to store application data such as Master and Transactional Data. We always create Transparent tables only.

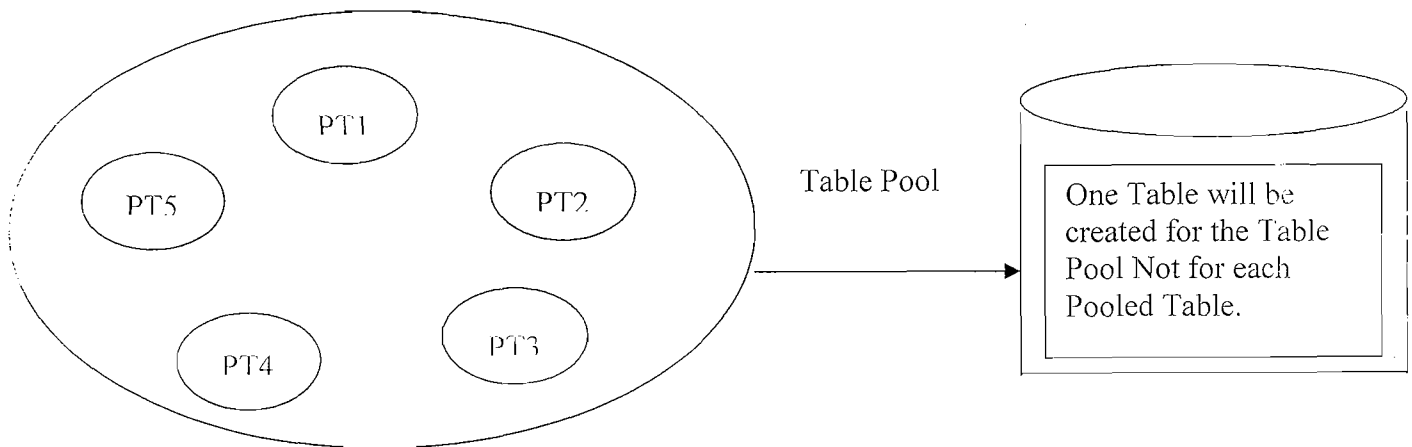
There is a One to One relationship i.e. for the table in DDIC another table with the same structure and the same name and the same fields will be created in the original Data base.

Pooled Tables:

It should be used exclusively for storing internal control information (screen sequences, program parameters, temporary data, continuous texts such as documentation).

The data from several different pooled tables can be stored together in a table pool.

This is Many to One relationship i.e. for many pooled tables in DDIC only one Data base table will be created in the Data Base.

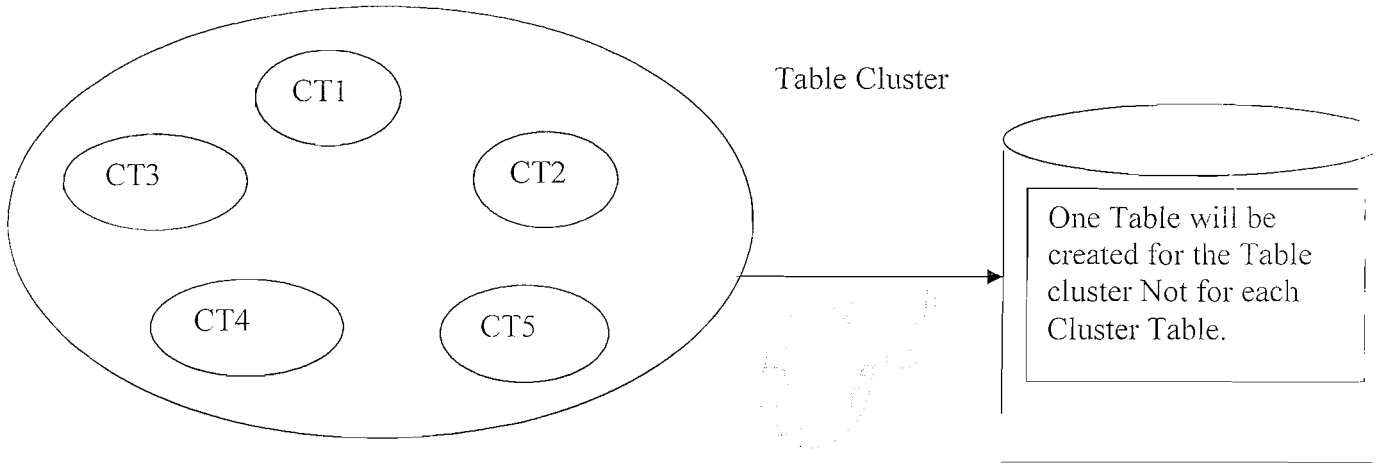


Cluster Table:-

It should be used exclusively for storing internal control information (screen sequences, program parameters, temporary data, continuous texts such as documentation).

The data from several different cluster tables can be stored together in a table clusters.

This is Many to One relationship i.e. for many cluster tables in DDIC only one Data base table will be created in the Data Base.



Note:- All the Tables in the Cluster Tables should have a common Primary Key.

In order to create a custom(userdefined) database table there are **TWO** types of approaches.

Exercises

1. Create database table with the following requirements:
(Using Direct Data Type)

Field Description	Data type	Length
Vendor Number	Char	10
Bank Country Key	Char	3
Bank Key	Char	06
Bank Acc. No	Char	12
Name of the holder	Char	35

2. Create the above table using data element and domains.
3. Create a database tables using the following fields, make use of INCLUDE structure for the common fields.

Table 1	Table 2	Table 3
LIFNR (Vendor No)	KUNNR (Customer No)	BUKRS (Company Code)
NAME1(Name)	NAME1(Name)	NAME1-Name
ORT01(City)	ORT01(City)	ORT01-City
ORT02(District)	ORT02(District)	ORT02-District
STRAS(Street)	STRAS(Street)	STRAS-Street
LAND1-Country	LAND1-Country	LAND1-Country
		CHAR,10
		CHAR,35
		CHAR,35
		CHAR,35
		CHAR,40
		CHAR,3

4. Add the below fields to the standard database table T001 by appending the following fields,

Fields	Data Element	Description
Website	CHAR,30	Website for the company
CEO	CHAR,50	CEO of the company

Create the table to maintain the Vendor Master Details :

Fields	Domain Details	Description
LIFNR	CHAR,10	Vendor Number
NAME1	CHAR,35	Vendor Name
ORT01	CHAR,35	Vendor City
ORT02	CHAR,30	District
STRAS	CHAR,30	Street

Note: LIFNR as the Primary Key.

Create the below table to Maintain the Purchase Order Details of the Vendors:

Fields	Data Element	Description
LIFNR	CHAR,10	Vendor No
EBELN	CHAR,10	Purchase Order No
NETWR	CURR,13	Total Price
WAERS	CUKY,5	Currency Key
MENGE	QUAN,13	Quantity
MEINS	UNIT,3	Unit Of Measurement

Note: LIFNR and EBELN Combination is the Primary Key.

Note: Maintain the foreign key relationship between the above two tables Based on the Vendor No.

5. **Create a projection view for the database table T001.**
6. **Create a database view for the database tables KNA1, KNB1.**

3.Introduction to ABAP/4?

Duration – 1 Day (* 2 Hrs)

A. ABAP Statements

B. Statements & Key words

C. ABAP Data Types

**D. Introduction to First ABAP
program**

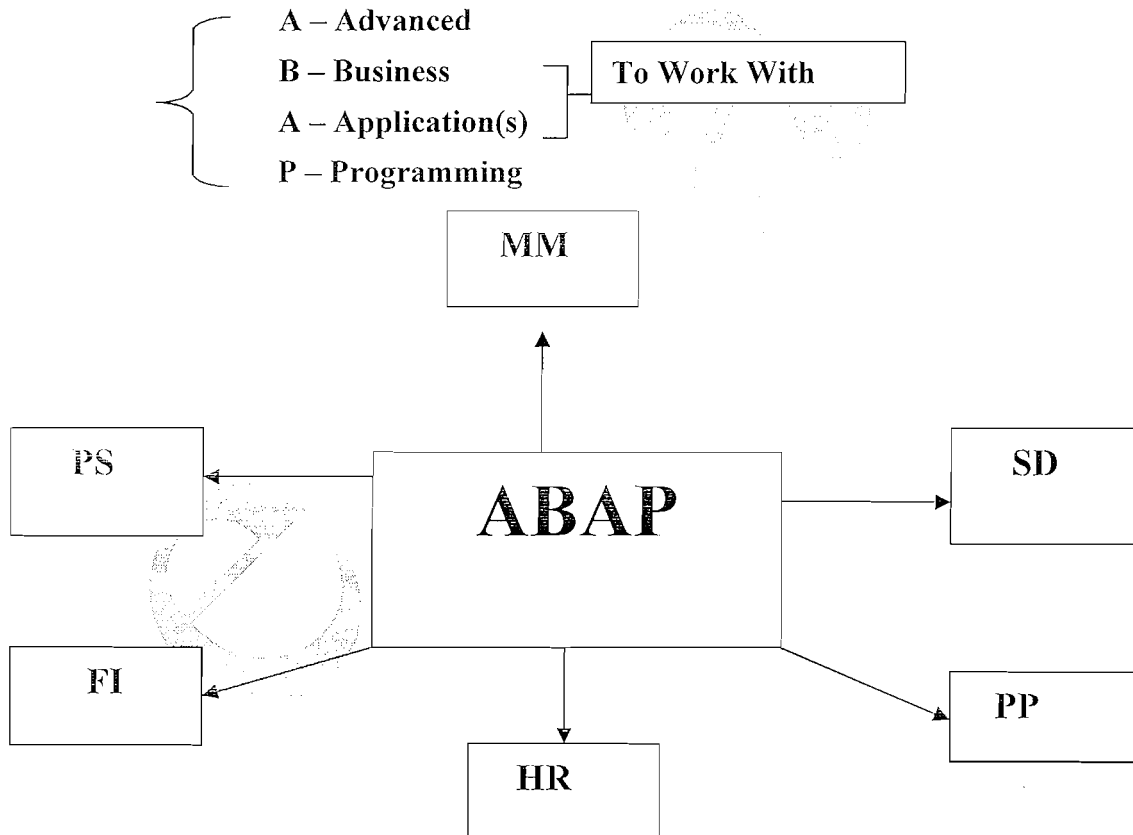
ABAP/4 – Advanced Business Application Programming language.

4 – 4th Generation Language

It is Not Case Sensitive.

Is Event Driven Programming Language.

This overview describes application programming in the R/3 System. All application programs, along with parts of the R/3 Basis system, are written in the ABAP Workbench using ABAP, SAP's programming language. The individual components of application programs are stored in a special section of the database called the R/3 Repository. The R/3 Repository serves as a central store for all of the development objects in the R/3 System. The following sections of this documentation cover the basics and characteristics of application programming.



Program:

A Program Is Group Of Meaningful Instructions.

An Instruction is Group Of

Keywords + Variables + Operators + Data types.

Keywords :

Syntax : Each ABAP statement Should begin with a keyword and ends with a period .

Since Each Statement Should Start with a Keyword , it is Difficult to give the exact no Of Keywords So that Keywords are Devided into Different Types Depends On the Functionality Of the Keywords.

Types Of Keywords:

Declarative Key words: To Declare Variables.

TYPES, DATA, TABLES

Syntax for Variable: DATA <Var.Name> TYPE <Data Type>.

Database Key words : To Work With Database Operations Such as

SELECT -- To Select Data

INSERT -- To Insert Data

UPDATE - To Change Data

DELETE - To Delete Data etc..

Control Key words

Statements are used to control the flow of an ABAP program within a processing block according to certain conditions.

Ex:

IF, ELSEIF, ENDIF.

DO-ENDDO, WHILE – ENDWHILE.

Definition keywords Are used to define Re-usable Modules(Blocks)

Ex:

FORM - ENDFORM

FUNCTION - ENDFUNCTION

MODULE - ENDMODULE.

Calling Keywords

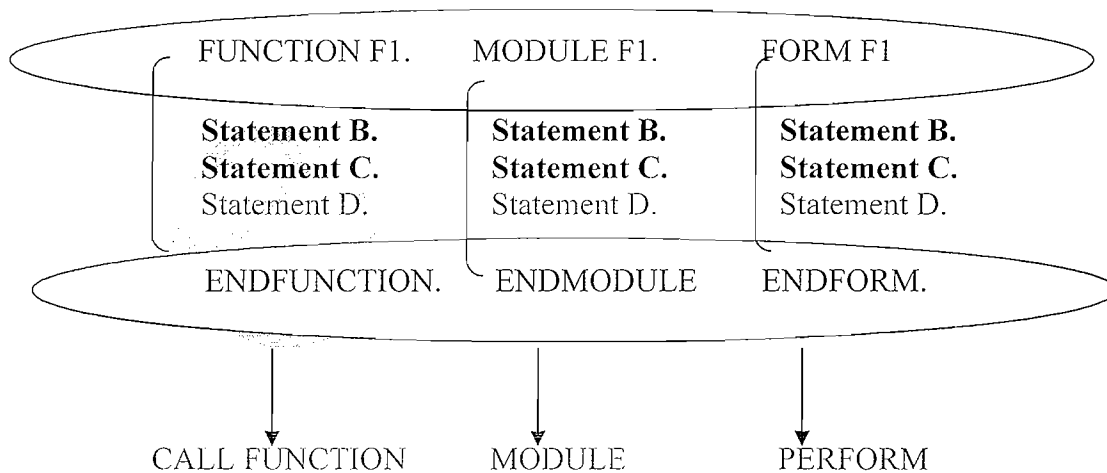
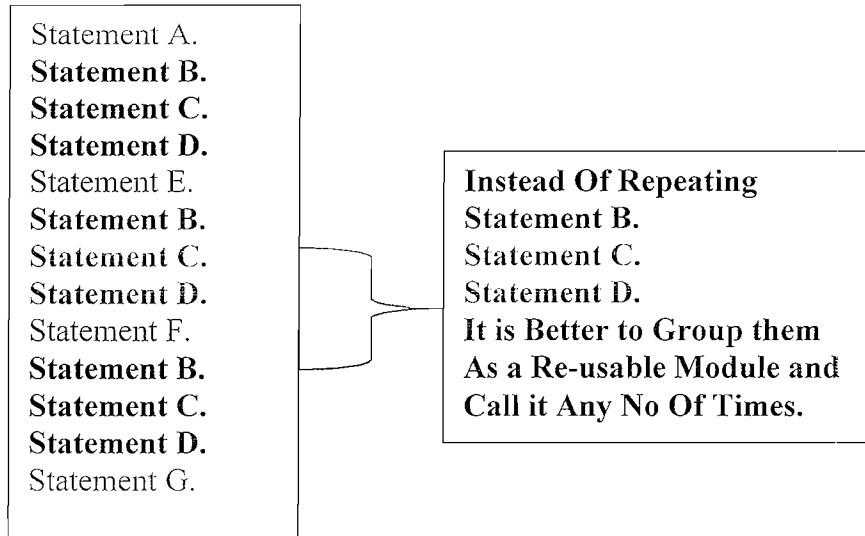
Are used to call Re-usable Modules(Blocks) that are already defined.

PERFORM to Call **FORM**

CALL FUNCTION to Call **FUNCTION**

MODULE to Call **MODULE**

Sample Program :



Operational Key words :

To Process the data that you have defined using declarative statements.

Ex : WRITE, MOVE, ADD

Event Keywords

Statements containing these keywords are used to define event blocks.

Ex: TOP-OF-PAGE To Print the Same Heading On the TOP Of Every Page

END-OF-PAGE. To Print the Same FOOTer for every Page Of the Output List.

Data Types and Objects (Variables)

The physical units with which ABAP statements work at runtime are called internal program data objects. The contents of a data object occupy memory space in the program. ABAP statements access these contents by addressing the name of the data object. Each ABAP data object has a set of technical attributes, which are fully defined at all times when an ABAP program is running. The technical attributes of a data object are: **Data type, field length, and number of decimal places.**

ABAP contains the following Pre-defined Data types:

Non-Numeric Data types:

Character string (C),

Numeric character string (N),

Date (D),

Time (T).

Numeric types:

Integer (I),

Floating-point number (F)

Packed number (P).

The **field length** for data types D, F, I, and T is fixed. The field length determines the number of bytes that the data object occupies in memory. **In types C, N, X and P, the length is not part of the type definition. Instead, you define it when you declare the data object in your program.**

Data type P is particularly useful for exact calculations in a business context. When you define an object with type P, you also specify a number of **decimal places.**

You can also define your own elementary data types in ABAP using the **TYPES** statement. You base these on the predefined data types. This determines all of the technical attributes of the new data type. For example, you could define a data type P_2 with two decimal places, based on the predefined data type P. You could then use this new type in your data declarations.

Predefine Elementary ABAP Types: All field lengths are specified in bytes.

Data Type	Initial Field length	Valid field length	Initial value	Meaning
Numeric types				
I	4	4	0	Integer (whole number)
F	8	8	0	Floating point number
P	8	1 - 16	0	Packed number
Character(Non-Numeric) types				
C	1	1 - 6553 5	'...'	Text field (alphanumeric characters)
D	8	8	'00000000'	Date field (Format: YYYYMMDD)
N	1	1 - 6553 5	'0 ... 0'	Numeric text field (numeric characters)
T	6	6	'000000'	Time field (format: HHMMSS)

Note : Data type N is **not** a numeric type. Type N objects can only contain **numeric characters** (0..9), but are not represented internally as numbers. Typical type N fields are account numbers and zip codes.

Integers - type I

The value range of type I numbers is -2^{31} to $2^{31}-1$ and includes only **whole numbers**. Non-integer results of arithmetic operations (e.g. fractions) are rounded, not truncated.

You can use type I data for counters, numbers of items, indexes, time periods, and so on.

Packed numbers - type P

Type P data allows digits after the decimal point. **The number of decimal places is generic, and is determined in the program.** The value range of type P data depends on its size and the number of digits after the decimal point. **The valid size can be any value from 1 to 16 bytes.** Two decimal digits are **packed** into one byte, while the last byte contains one digit and the sign. Up to 14 digits are allowed after the decimal point. The initial value is zero. **When working with type P data, it is a good idea to set the program attributes *Fixed point arithmetic*.** **Otherwise, type P numbers are treated as integers.**

Note: You can use type P data for such values as distances, weights, amounts of money, and so on.

Floating Point numbers - type F

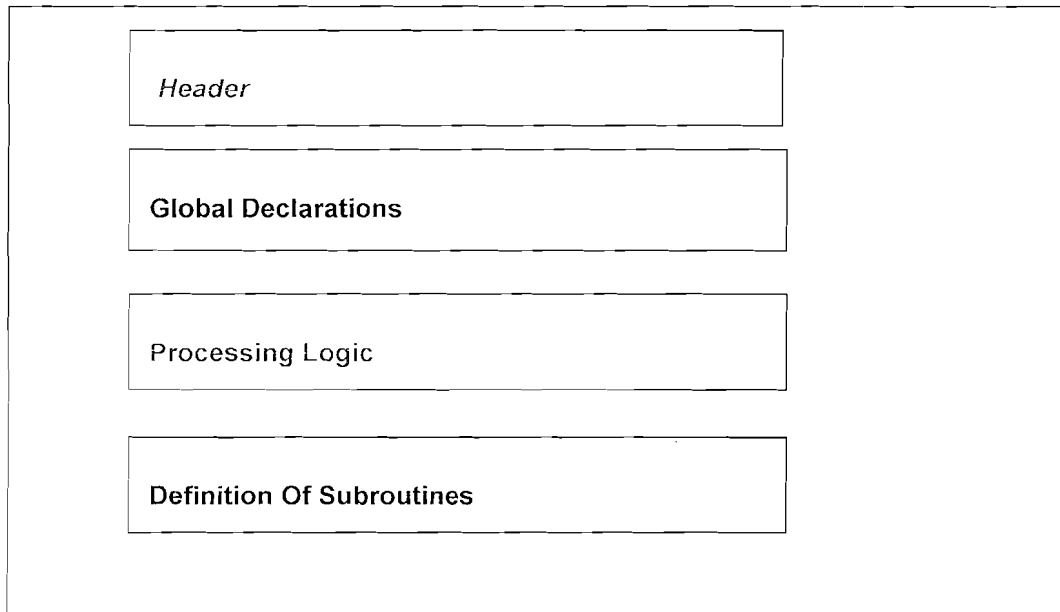
The value range of type F numbers is **1×10^{-307} to 1×10^{308}** for positive and negative numbers, including 0 (zero). The accuracy range is approximately 15 decimals, depending on the **floating point** arithmetic of the hardware platform. Since type F data is internally converted to a binary system, rounding errors can occur. **Although the ABAP processor tries to minimize these effects, you should not use type F data if high accuracy is required. Instead, use type P data.**

You use type F fields when you need to cope with very large value ranges and rounding errors are not critical.

Using I and F fields for calculations is quicker than using P fields. Arithmetic operations using I and F fields are very similar to the actual machine code operations, while P fields require more support from the software. Nevertheless, you have to use type P data to meet accuracy or value range requirements.

Character(Non-Numeric) types

Of the five **non-numeric** types, the four types C, D, N, and T are **character types**. Fields with these types are known as **character fields**. Each **position** in one of these fields takes up enough space for the code of one character. Currently, ABAP only works with single-byte codes such as ASCII and EBCDI. However, an adaptation to UNICODE is in preparation. Under UNICODE, each character occupies two or four bytes.

Structure Of ABAP Program :

Header Section: Is to provide More Information about the Development and which is a Standard Template for all the Custom ABAP Developments.

```

*****
* PROGRAM          : ZDEMO_HELLO_ABAP
* AUTHOR           : GANAPATI.ADIMULAM
* START DATE      : 26/01/2008
* PURPOSE         : IS TO PRINT THE DETAILS OF ABAP
* COPIED FROM     : NA
* MODIFICATION LOG:
* MOD-001         : DETAILS OF MODIFICATION 001
* MOD-002         : DETAILS OF MODIFICATION 002
* SUPPLIERS       : EMAX TECHNOLOGIES
*****
  
```

Global Declaration Block is to Declare all the Global Variables.

Processing Logic Block is to Implement the Business Logic.

Definition Of Subroutines :

Subroutines are the reusable Components . We Define the reusable Components Once and We call the same subroutine wherever we need the same Business Logic.

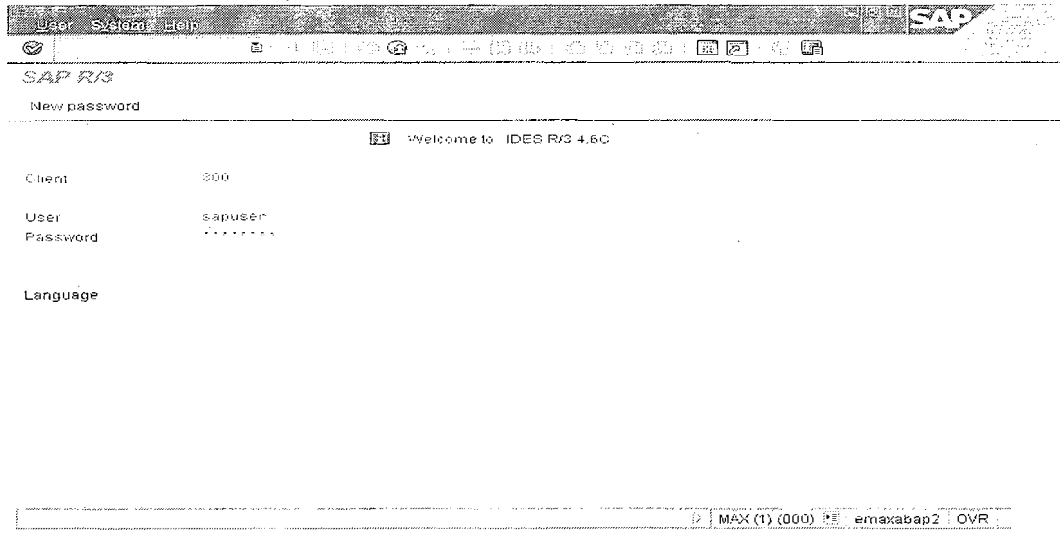
Note: No Executable Statements Can be Accessed after the Definition Of Subroutine Definitions.

Introduction to Create First ABAP Program:

DAY-2

1. Log On screen

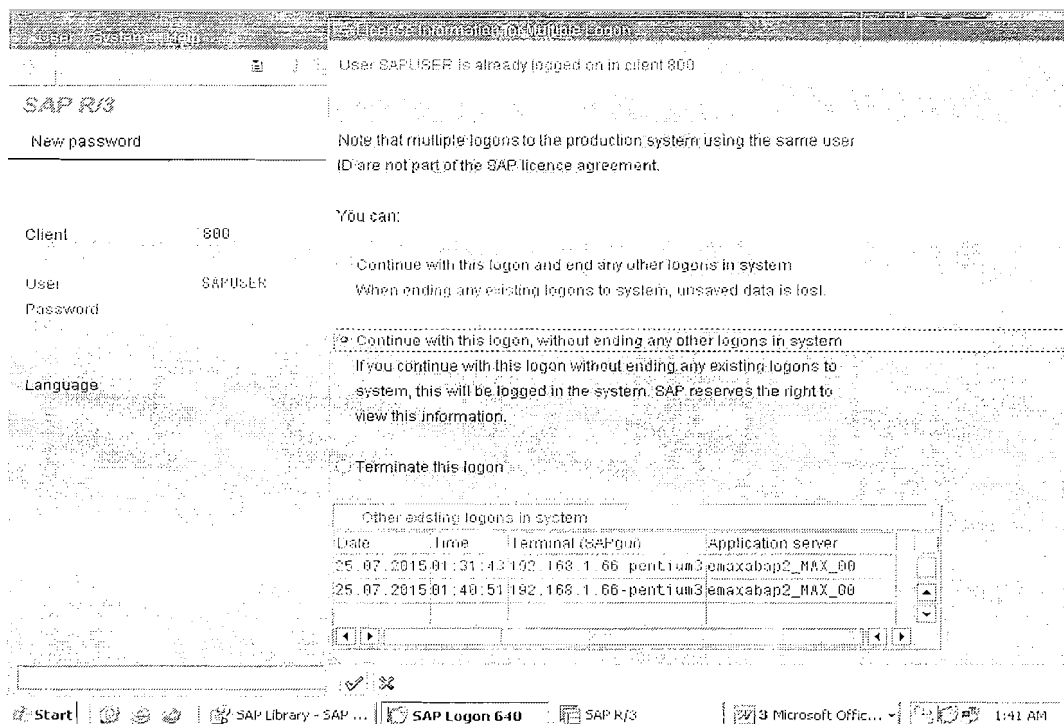
Provide the Client ID, User ID and Password




Client: - Is an independent organizational unit which is having its own Master data and Tables.

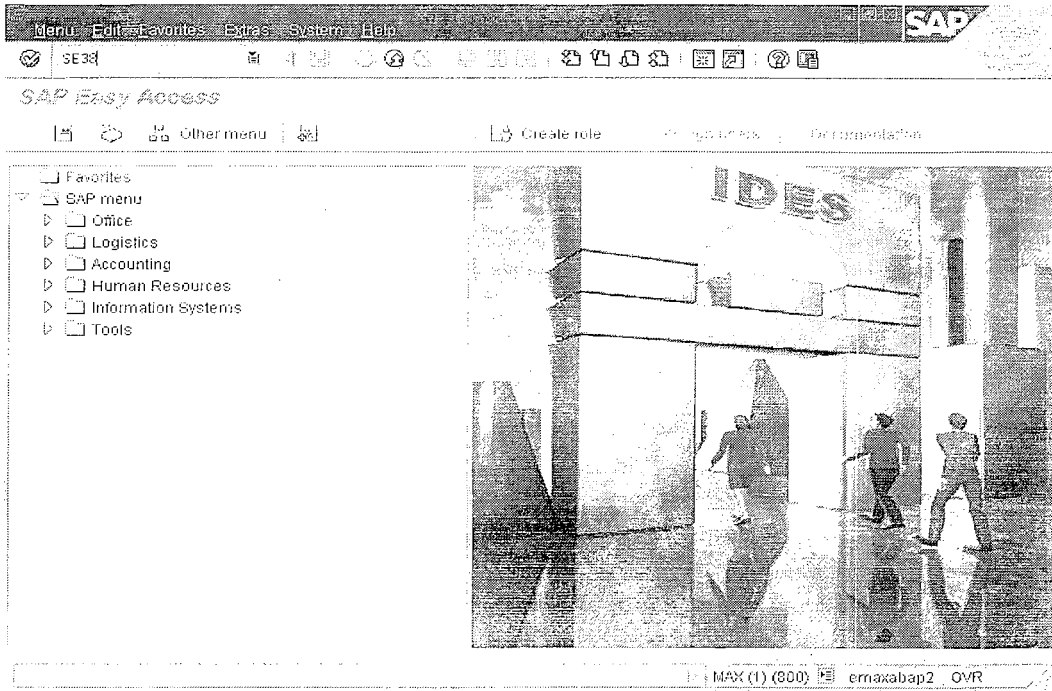
Log on credentials will be provided by the customer , the Details are Unique for Each Developer except Client in **REAL TIME**..

The following screen pops up. Select the middle radio button as follows.

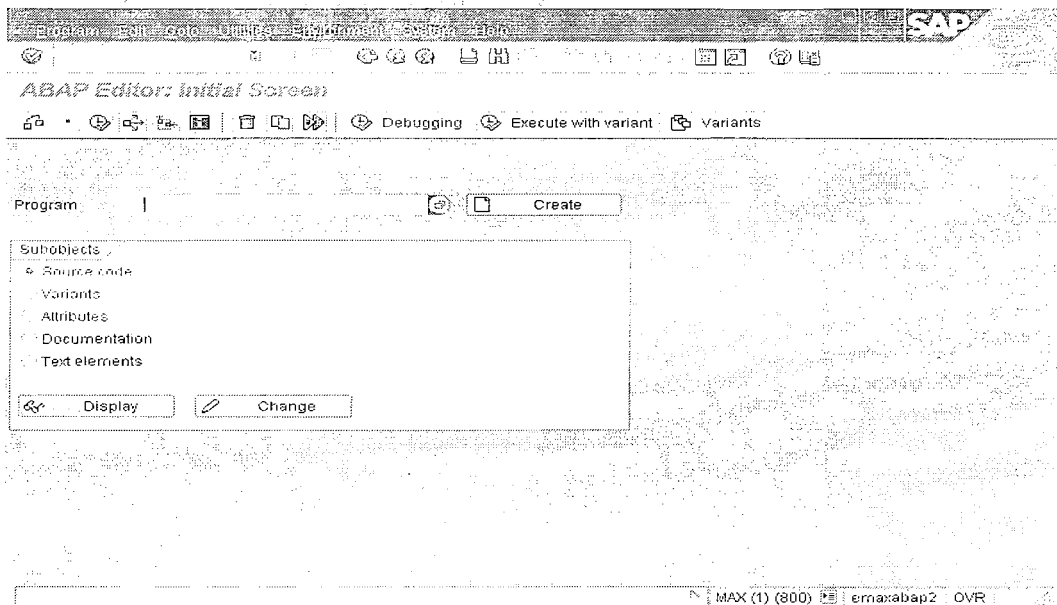


2. SE38 is the Transaction code for ABAP Editor.


Type SE38 in the Transaction bar. 



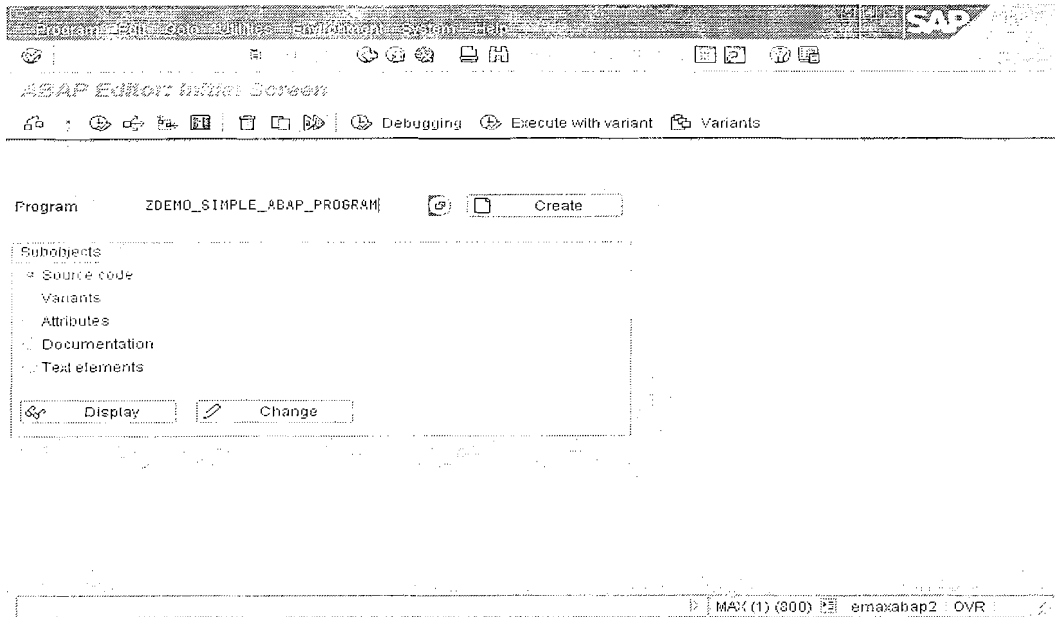
3. The ABAP Editor screen opens up as follows



4. Enter the Program name in the box and

Select the Source Code  in the sub-objects and

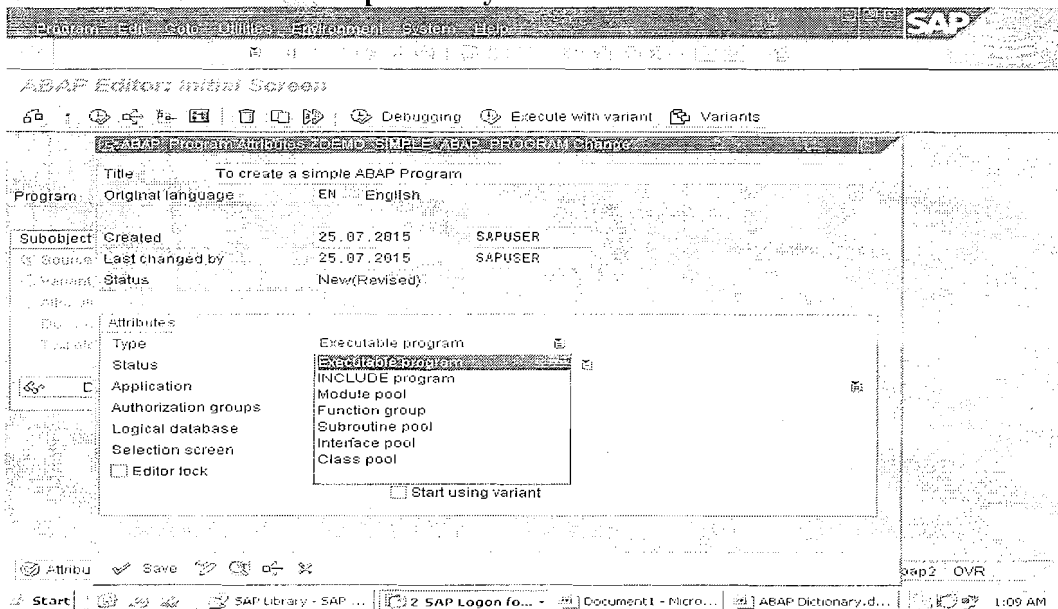
Press 'CREATE'  Create button




NOTE : It is Mandatory to start Program name with 'Z' or 'Y' as 'A' to 'X' is Reserved for SAP .

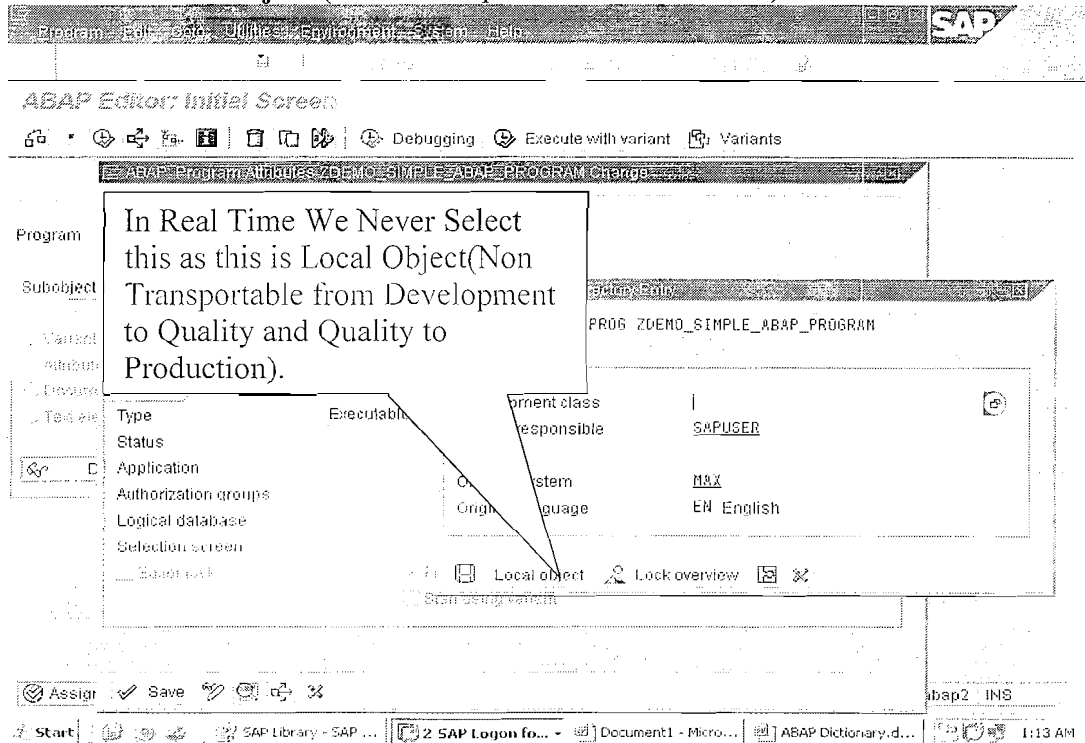
Note : In Real time, the Custom Development Names Can also starts with other than 'Y' and 'Z' i.e /<Any 8 Alphanumerics>/

5. Provide the Title and Type of the Program as 'Executable Program', Which Can be tested Independently.

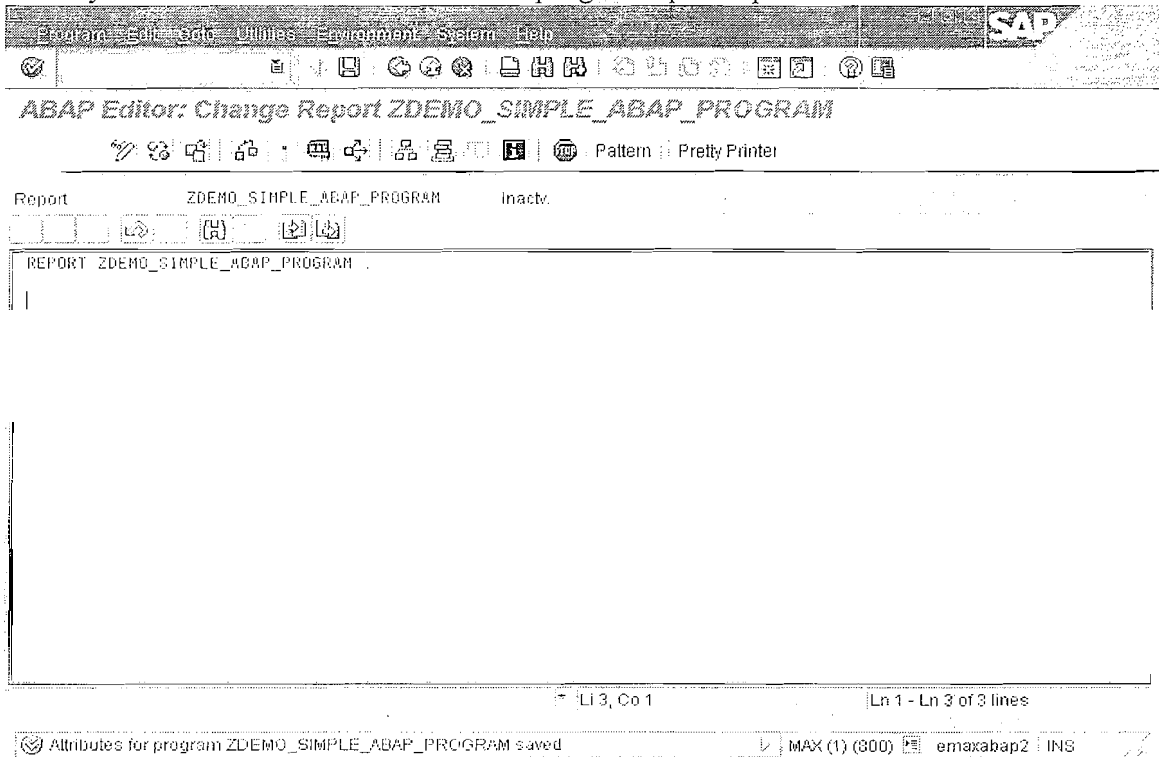


SAVE .  Save or ENTER.

Click On **Local Object** (The Development Class is &TMP)



Finally the ABAP Editor for the current program opens up as follows.

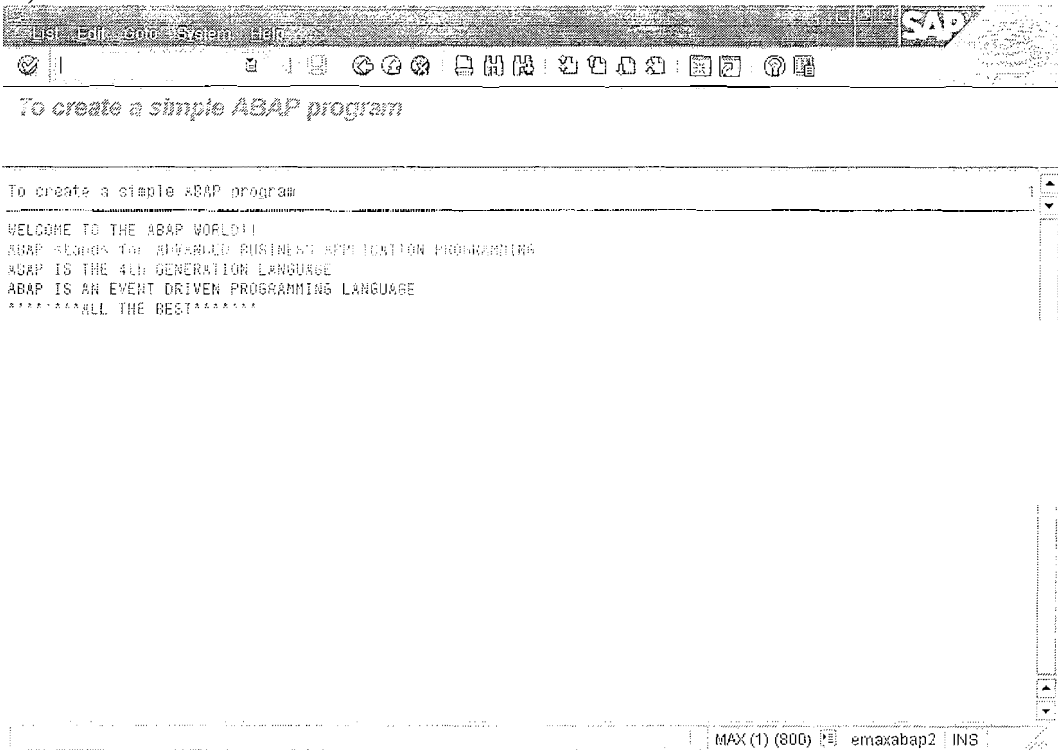


Press **ENTER** or .

Now press the **TEST** button  OR press (F8) for output screen.



3.) The **OUTPUT SCREEN** is as follows.



Working With Arithmetic Operators using PARAMETER:

Note : PARAMETER is to accept the Input via Console (Screen) , Which Acts as Scanf in 'C'.

```
REPORT ZGDEMO_PARAMETER
```

```
PARAMETER : P_INPUT1 TYPE I,
            P_INPUT2 TYPE I.
```

```
DATA V_RESULT TYPE I.
```

```
V_RESULT = P_INPUT1 + P_INPUT2.
```

```
WRITE : / 'THE RESULT OF ADDITION IS', V_RESULT.
```

```
V_RESULT = P_INPUT1 * P_INPUT2.
```

```
WRITE : / 'THE RESULT OF MULTIPLICATION IS', V_RESULT.
```

```
V_RESULT = P_INPUT1 MOD P_INPUT2.
```

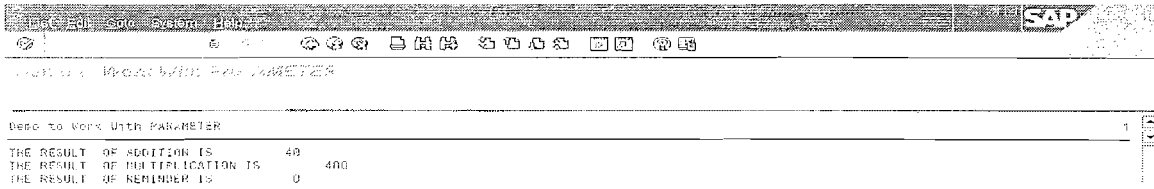
```
WRITE : / 'THE RESULT OF REMINDER IS', V_RESULT.
```

OUTPUT :**EXECUTE THE PROGRAM**

```

P_INPUT1      20
P_INPUT2      26
  
```

Provide the INPUT and EXECUTE it.

OUTPUT


```

-----
DEMO TO WORK WITH PARAMETER
-----
THE RESULT OF ADDITION IS      40
THE RESULT OF MULTIPLICATION IS 400
THE RESULT OF REMINDER IS      0
-----

```

DEV (2) (000) gsnapathi INS

WORKING WITH SYSTEM VARIABLES:

SYSTEM VARIABLE: A VARIABLE, Which is Declared and Filled by the SAP System.

Note : All the System Variables are Maintained in Database Structure SYST.
Accessing System Variable is SY-<Name>.

```

*&-----*
*& Report  ZGDEMO_SYSTEM_VARIABLES
*&
*&-----*

REPORT  ZGDEMO_SYSTEM_VARIABLES

WRITE /40 'THE RESULT OF SOME SYSTEM VARIABLES'.
ULINE.

WRITE : /40 'CURRENT DATE IS', 60 SY-DATUM.
WRITE : /40 'CURRENT TIME IS', 60 SY-UZEIT.
WRITE : /40 'CURRENT USER IS', 60 SY-UNAME.
WRITE : /40 'CURRENT PROG.NAME ', 60 SY-REPID.
WRITE : /40 'TITLE IS',          60 SY-TITLE

```

OUTPUT : EXECUTE THE PROGRAM

```

DEMO ON SYSTEM VARIABLES
-----
BY RESULT OF USED SYSTEM VARIABLES
-----
CURRENT DATE IS      16.04.2011
CURRENT TIME IS     16:46:01
CURRENT USER IS     SAPUSER
CURRENT PROGRAM IS  ZGDEMO_SYSTEM_VARIABLES
TITLE IS            Demo On System Variables
-----

```

WORKING WITH CHAIN OPERATOR (:)

To Group(Chain) Sequential Statements Which Starts with the Same Keyword(s).

```

*&-----*
*& Report  ZGDEMO_CHAIN_OPERATOR *
*& *
*&-----*

REPORT  ZGDEMO_CHAIN_OPERATOR

*Without Chain Operator
WRITE / 'RESULT USING CHAIN OPERATOR'.
ULINE.
WRITE / 'Hello ABAP'.
WRITE / 'Welcome to eMAX Technologies'.
WRITE / 'All SAP Modules are Developed by ABAP'.
WRITE / 'SAP is from SAP AG'.

SKIP 2.

*With Chain Operator
WRITE / 'RESULT WITHOUT CHAIN OPERATOR'.
ULINE.
WRITE / : 'Hello ABAP',
          'Welcome to eMAX Technologies',
          'All SAP Modules are Developed by ABAP',
          'SAP is from SAP AG'.

```

OUTPUT : EXECUTE PROGRAM ZGDEMO_CHAIN_OPERATOR

DEV (2) (8000) ganapati@INS



Exercises

1. Write a Program to Print the below

27/12/2006

Hello ABAP Program

The List of System Variables are :

Current Date : 27/12/2006
Current Time : 13:40:35
User Name : SAPUSER
Database : ORACLE
Program Name : ZDEMO_HELLO_ABAP.

From eMAX Technologies

2. Write a program to Display the List of Modules in SAP.

List of Modules in SAP are

MM - Materials Management
SD - Sales & Distribution
PP - Production Planning
HR - Human Resources
FI&CO - Finance & Control
CRM - Customer Relationship Management
SEM - Strategic Enterprise Management

From eMAX Technologies

3. Write a Program to Print the List of Important Days in a Year ?

List of Important Days in Year

1 st January	-	New Year
14 th January	-	Sankranthi
26 th January	-	Republic Day
14 th February	-	Valentine Day
.		
.		
2 nd October	-	Gandhi Jayanthi
14 th November	-	Children's Day
6 th December	-	eMAX Anniversary Day
25 th Decemeber	-	Christmas

From eMAX Technogies

4. Write a Program to Print the List of your Best Friends in the Below Format.

Best Friends Directory

Name	Gender	Date of Birth
Vijaya Gouri	Female	10/07/1978
GVT Balaji	Male	25/12/1981
Mubeena	Female	28/10/1982
Kaadambari	Male	30/08/1982
Srinivas	Male	16/05/1983
Murali	Male	24/06/1983

From eMAX Technogies

5. Show a value '123456' as 12:34:56 using Edit Mask.
6. Take a number as '0000011', and Suppress all leading zeros.
7. Suppress a sign before a number.
8. Accept two values from selection screen and perform the following :
+, -, *, /, **, mod and display the Result.

4. Control Structures

Duration – 1 Day (* 2 Hrs)

A. BRANCHING

1. IF-ELSE/ ELSEIF-
ENDIF
2. CASE-WHEN –
ENDCASE

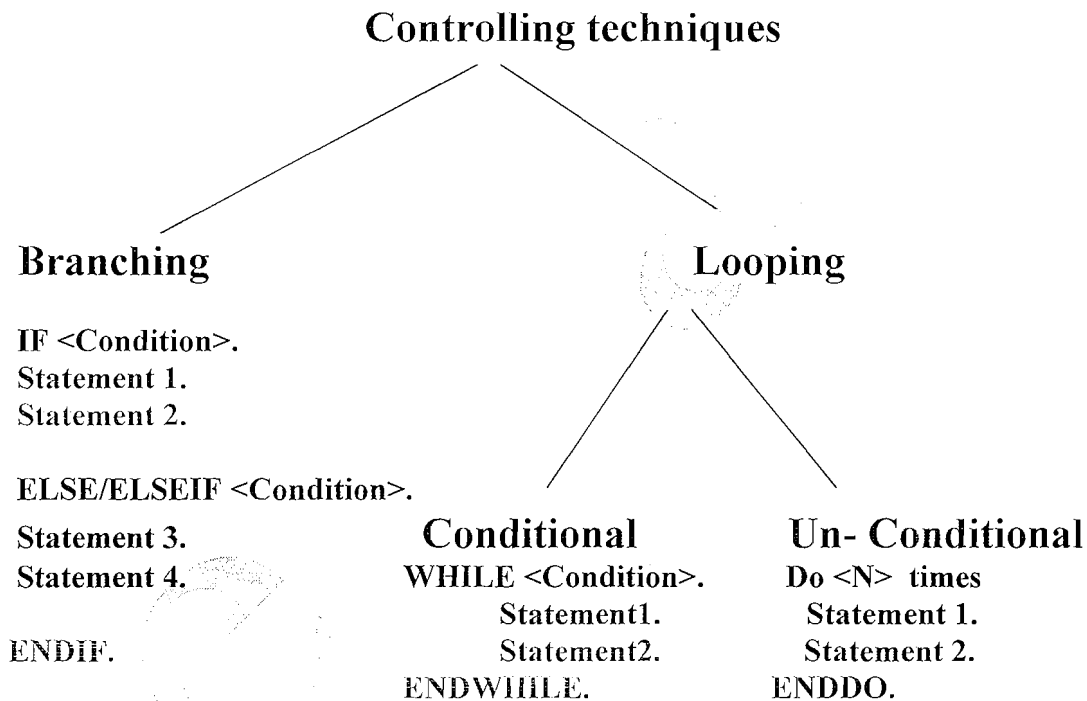
B. LOOPING

1. Conditional (DO-
ENDDO)
2. Un-Conditional
(WHILE-
ENDWHILE)

Note: The Program execution is always Sequential. The First Statement is Executed First and the Nth Statement will be after (N-1) the statement and Each Statement is Executed Only Once by DEFAULT .

But Most of the times we need to CONTROL the traditional FLOW Of Program Execution BY branching from One Block of statements to another block depends on the condition and have to repeat a particular block of statements THROUGH **Controlling Techniques**.

You can execute different parts of programs conditionally or in loops using the standard keywords **IF, CASE, DO, and WHILE**.



Alternative for IF-ELSEIF-ENDIF is CASE-ENDCASE.

IF-ELSEIF-ENDIF

This control structure is introduced with the IF statement. The IF statement allows you to divert the program flow to a particular statement block, depending on a condition. The statement block concludes either with ENDIF, ELSEIF, or ELSE.

```

IF <condition1>.
  <statement block>
ELSEIF <condition2>
  <statement block>.
  
```

```

ELSEIF <condition3>.
    <statement block>
.....
ELSE.
    <statement block>
ENDIF.

```

If the first condition is true, the system executes all the statements up to the end of the first statement block and then continues processing after the ENDIF statement. If the first condition is not true, the program jumps to the next ELSEIF statement and executes it like an IF statement. ELSE begins a statement block which is processed if none of the IF or ELSEIF conditions is true. The end of the last statement block must always be concluded by ENDIF.

The CASE Control Structure : This control structure is introduced with the CASE statement. The CASE control structure allows you to control which statement blocks are processed based on the contents of a data object.

```

CASE <f>.
    WHEN <f1>.
        <Statement block>
    WHEN <f2>.
        <Statement block>
    WHEN <f3>.
        <statement block>
    WHEN ...
    .....
    WHEN OTHERS.
        <statement block>
ENDCASE.

```

EXAMPLE PROGRAM ON IF-ELSEIF-ENDIF:

```

*&-----*
*& PROGRAM           :      ZGDEMO_IF_ELSEIF_ENDIF          *
** AUTHOR            :      GANAPATI . ADIMULAM            *
*& PURPOSE           :      WORKING WITH IF-ELSEIF-ENDIF    *
*&-----*

REPORT  ZGDEMO_IF_ELSEIF_ENDIF

PARAMETER P_DAY TYPE I.

IF P_DAY = 1.
    WRITE / 'SUNDAY'.
ELSEIF P_DAY = 1.
    WRITE / 'MONDAY'.
ELSEIF P_DAY = 2.
    WRITE / 'TUESDAY'.
ELSEIF P_DAY = 3.
    WRITE / 'WEDNESDAY'.

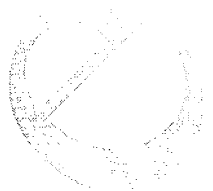
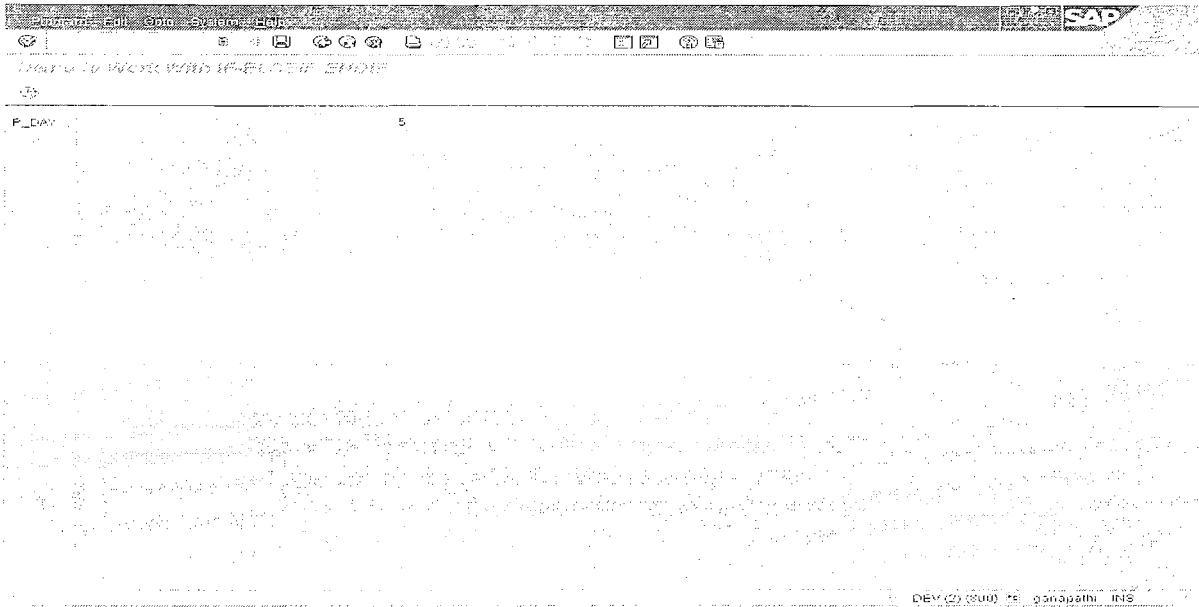
```

```
ELSEIF P_DAY = 4.  
    WRITE / 'THURSDAY'.  
ELSEIF P_DAY = 5.  
    WRITE / 'FRIDAY'.  
ELSEIF P_DAY = 6.  
    WRITE / 'SATURDAY'.  
ELSE.  
    WRITE / 'DAY SHOULD BE BETWEEN 0 AND 6 ONLY'.  
ENDIF.
```

OUTPUT

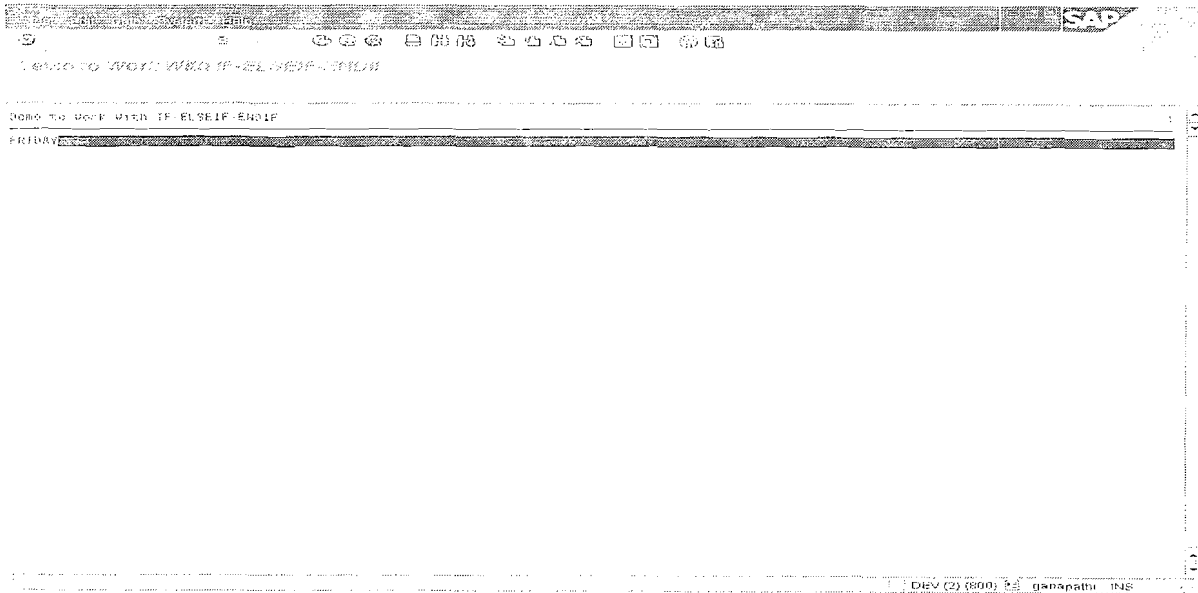
EXECUTE THE PROGRAM

CASE 1 :

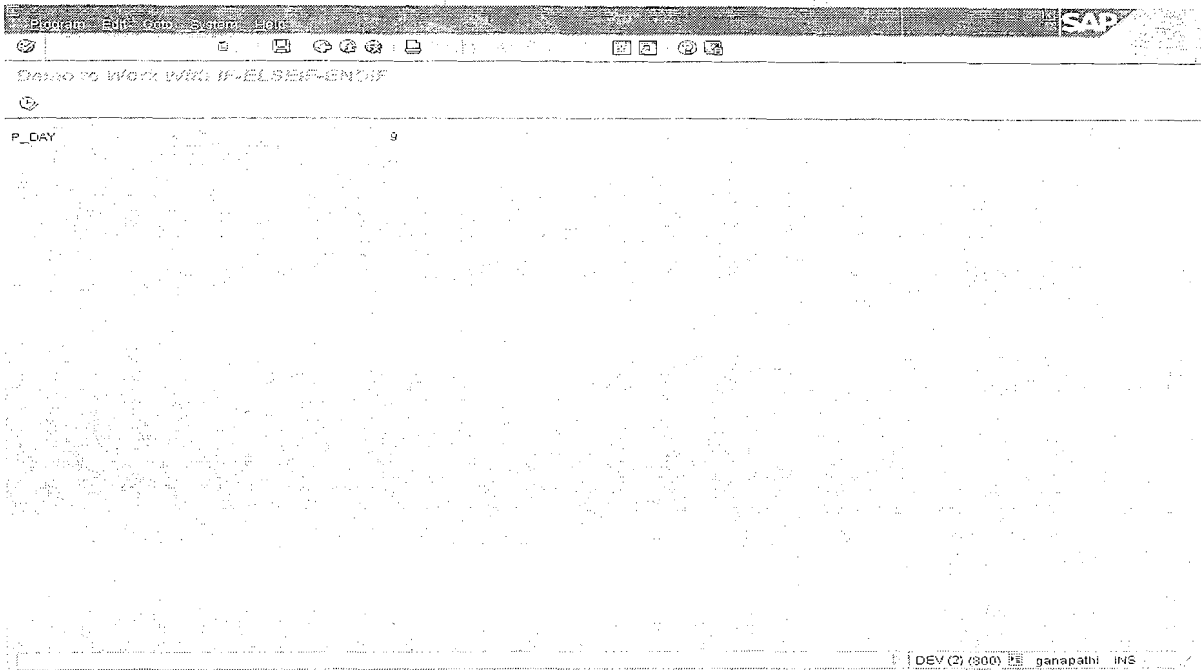


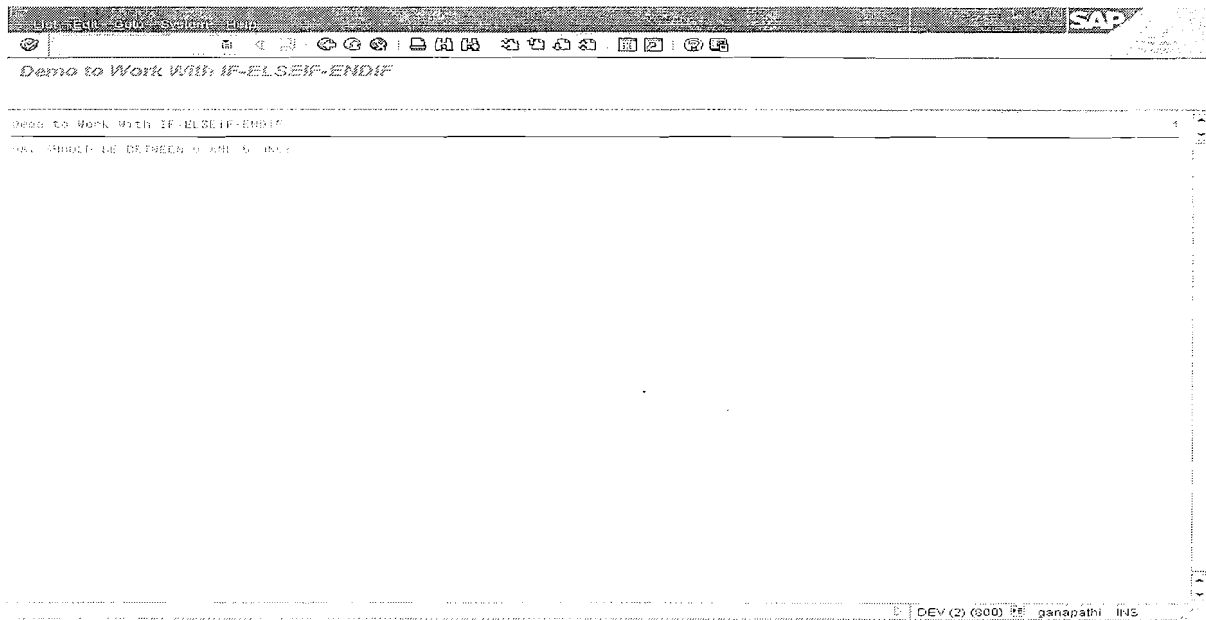
Control Structures

We Never Compromise in Quality, Would You?



CASE 2 :





EXMPLME PROGRAM ON CASE-WHEN-ENDCASE

```
*&-----*
*& PROGRAM          :      ZGDEMO_CASE_WHEN_ENDCASE          *
** AUTHOR           :      GANAPATI . ADIMULAM              *
*& PURPOSE          :      WORKING WITH CASE-WHEN-ENDCASE    *
*&-----*

```

```
REPORT ZGDEMO_CASE_WHEN_ENDCASE
```

```
PARAMETER P_DAY TYPE I.
```

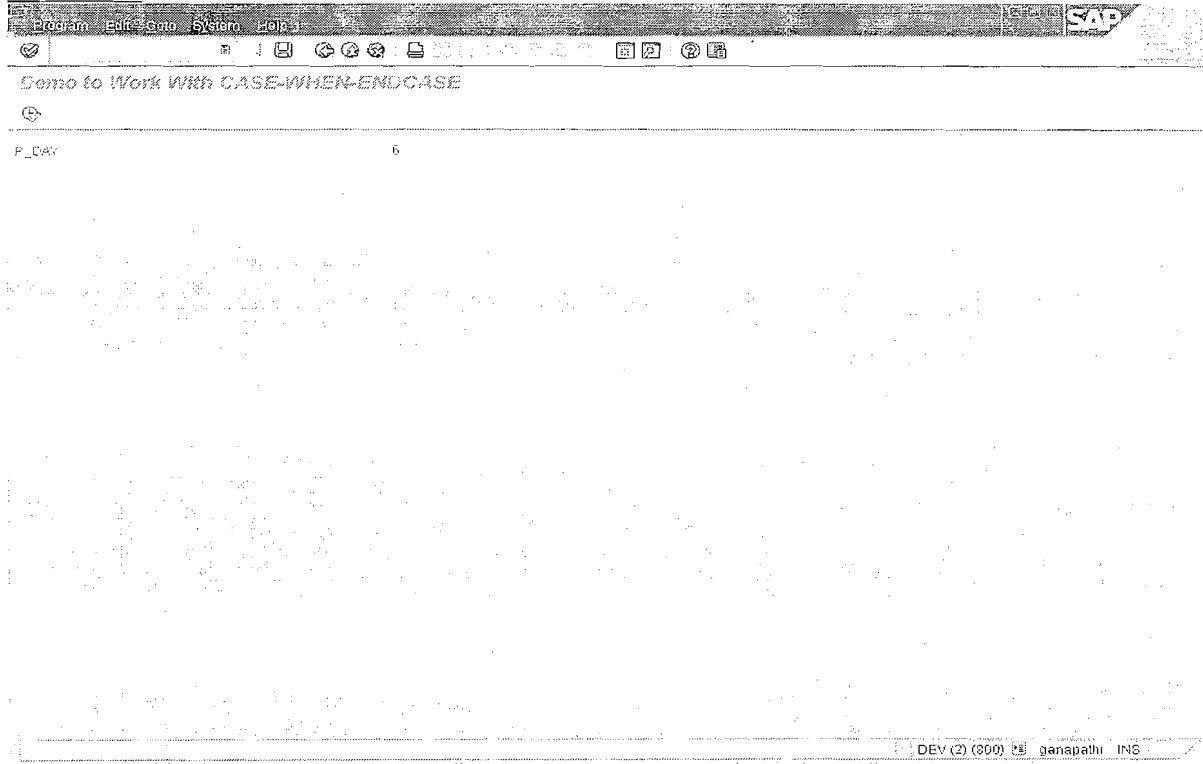
```
CASE P_DAY.
```

```
  WHEN 1.
    WRITE / 'SUNDAY'.
  WHEN 1.
    WRITE / 'MONDAY'.
  WHEN 2.
    WRITE / 'TUESDAY'.
  WHEN 3.
    WRITE / 'WEDNESDAY'.
  WHEN 4.
    WRITE / 'THURSDAY'.
  WHEN 5.
    WRITE / 'FRIDAY'.
  WHEN 6.
```

```
WRITE / 'SATURDAY'.  
WHEN OTHERS.  
WRITE / 'DAY SHOULD BE BETWEEN 0 AND 6 ONLY'.
```

ENDCASE.

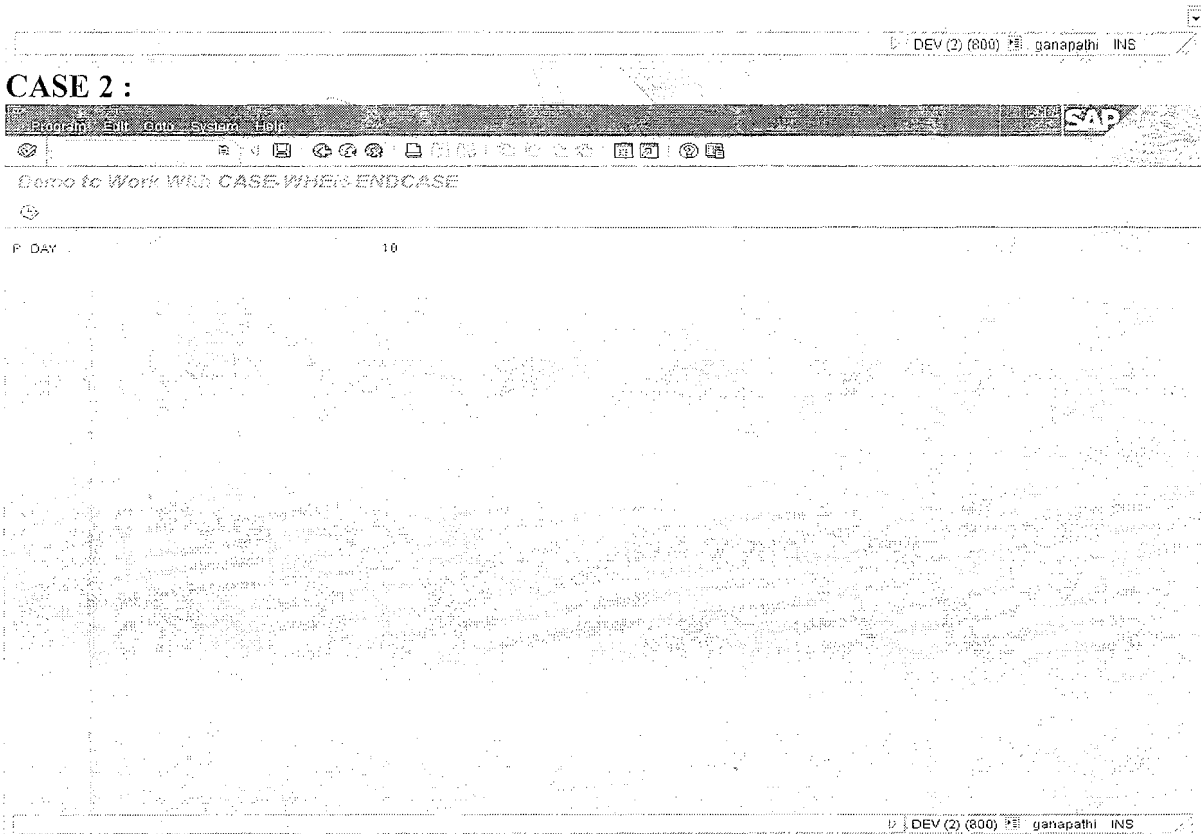
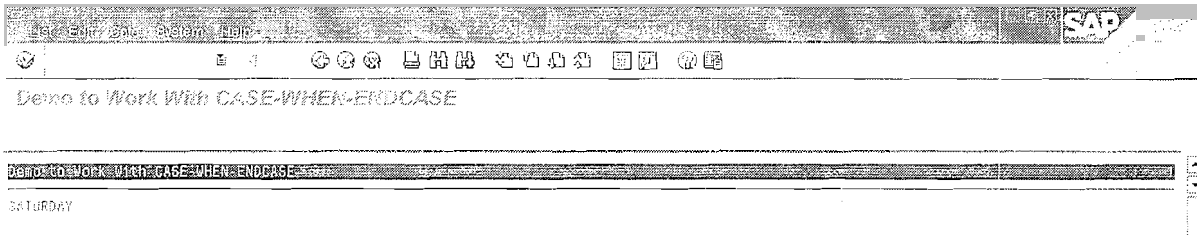
CASE 1 : EXECUTE PROGRAM



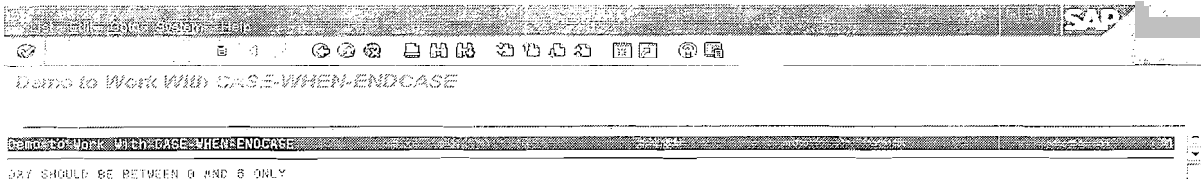
EXECUTE

Control Structures

We Never Compromise in Quality, Would You?



EXECUTE



Working With Looping :

Types Of Looping :

- A) Un Conditional.
- B) Conditional

Un Conditional: To Repeat a Loop for <N> No Of times. (N>0)

Syntax : DO <N> TIMES.

```
---  
---  
ENDDO.
```

Write a Program to Print First 10 Numbers.

```
DATA : V_COUNT TYPE I.  
DO 10 TIMES.  
    V_COUNT = V_COUNT + 1.  
    Write : / V_COUNT.  
ENDDO.
```

Output :

1

2
3
4
5
6
7
8
9
10

Note: SY-INDEX is the System Variable which keeps track of the Loop Counter. i.e the value of SY-INDEX Becomes 1 when it enters into the LOOP for the First time, 2 for 2nd time and similarly every time it enters into LOOP, SY-INDEX is Incremented By 1.

Example Of Nested Un Conditional Loops :

```
DO 2 TIMES.  
  WRITE SY-INDEX.  
  SKIP.  
  DO 3 TIMES.  
    WRITE SY-INDEX.  
  ENDDO.  
  SKIP.  
ENDDO.
```

The output is:

```
1  
1   2   3  
2  
1   2   3
```

Conditional Looping: Can be Looped based on the Condition. This loop will Continue the Condition is False.

SYNTAX: WHILE (Condition).

**Processing Block.
ENDWHILE.**

Example Prog :

```
WHILE sy-index <= 10.  
    Write : / sy-index.  
ENDWHILE.
```

Output :

```
1  
2  
3  
4  
5  
6  
7  
8  
9  
10
```

Terminating a Loop Pass Conditionally

To terminate a single loop pass conditionally, use the CHECK <condition> statement in the statement block of the loop.

If the condition is not true, any remaining statements in the current statement block after the CHECK statement are ignored, and the next loop pass starts. <condition> can be any logical expression.

```
DO 4 TIMES.  
    CHECK SY-INDEX BETWEEN 2 and 3.  
    WRITE SY-INDEX.  
ENDDO.
```

The output is:

```
2    3
```

The first and fourth loop passes are terminated without the WRITE statement being processed, because SY-INDEX is not between 2 and 3.

Exiting a Loop

To terminate an entire loop immediately and unconditionally, use the EXIT statement in the statement block of the loop.

After this statement, the loop is terminated, and processing resumes after the closing statement of the loop structure (ENDDO, ENDWHILE, ENDLOOP, ENDSELECT). In nested loops, only the current loop is terminated.

```
DO 4 TIMES.  
  IF SY-INDEX = 3.  
    EXIT.  
  ENDIF.  
  WRITE SY-INDEX.  
ENDDO.
```

The output is:

```
1      2
```

In the third loop pass, the loop is terminated before the WRITE statement is processed.

Exercise

1. Write a program to display the squares of first N natural Nos. where N is an Integer Input Parameter.
2. Write a program to accept a number (say 2) from user and create a multiplication table .
3. Write a Program to display the Fibonacci series below N. Where N is an Integer Input Parameter. ?
4. Write a Program to check whether the given number is prime or not.
5. Accept a number from user and find Factorial of the same ?
6. Write a Program to display list of first n even NOs, where n is the Input PARAMETER.
7. Write a Program to display the list of even and odd nos below the Input no(Integer PARAMETER) and Find out the SUM of Even Nos and also the SUM Of the ODD Nos and Check for the Equality of both the SUMs.
8. Create a calculator which performs four basic types of calculations on two whole numbers. The two values and the option to be entered on the selection screen as parameters. Output the result based on the Operator Entered.
9. Create your output as shown below

```
*  
**  
***  
****  
*****
```

And

```
1  
1 2  
1 2 3  
1 2 3 4  
1 2 3 4 5
```

10. Write a Program to accept the two number from the user and swap them.

5. String Operations

Duration – 1 Day (* 2 Hrs)

Working with Strings

String is a variable length data type.

Dynamic memory management is used internally. i.e the Memory Allocated(Adjusted) at runtime according to the current field content .

Strings can have any length(Can Store any no of Characters).

NOTE : Since STRING is a dynamic length data type, We Cannot Declare String Variables through **PARAMETER. i.e**

PRAMETER P_NAME TYPE STRING. Is Not Allowed. Because the system cannot understand how big the Input Field is.

CHAR	STRING
DATA V_NAME(100) TYPE C. V_NAME = 'eMAX'.	DATA V_NAME TYPE STRING. V_NAME = 'eMAX'.
Note : Memory is Always Allocated For 100 Chars IrrespectiveOf the No Of Chars Stored Currently.	Note : Memory is Allocated Only For eMAX , Which always depends On the No of Chars Stored Currently.

String Operations:

CONCATENATE : To Club More than One Sub String into One Main String.

SYNTAX : CONCATENATE <Str1> <Str2> <Str3> ... <Str N> INTO <Str>
SEPARATED BY <Separator>.

Ex: CONCATENATE 'E001' 'Emax Technologies' 'HYD' INTO V_NAME
SEPARATED BY ','.

Result: E001,eMAX Technologies,HYD.

CONDENSE: Replaces sequence of SPACES into exactly one SPACE.

CONDENSE 'Emax Technologies'.

Result :

Emax Technologies(All the Sequential Blanks Converted into Single Blank)

CONDENSE 'Emax Technologies' NO-GAPS.

Result:

EmaxTechnologies (All the Sequential Blanks Converted into Single Blank and even it deletes that single Space Also)

TRANSLATE: To Translate to UPPER/LOWER CASE.

TRANSLATE 'eMAX' TO UPPER CASE.

Result : EMAX.

TRANSLATE 'eMAX' TO LOWER CASE.

Result: exam.

REPLACE:

REPLACE <Str1> WITH <Str2> INTO <Str>.

Replaces **ONLY** the first occurrence (But **NOT ALL**) of the contents of field <Str1> WITH <Str2> INTO <Str>.

REPLACE 'e' WITH 'i' INTO 'eMAX Technologies' .

Result : 'iMAX Technologies' .

(Only the First Occurrence is replaced but not all the Occurrences).

SPLIT: To Split the Main String into Substrings at the given Seperator.

1. SPLIT <Str> AT <Sep> INTO <Str1> <Str2> ... <Str n> .

Ex : SPLIT 'E001,Emax Technologies,Ameerpet,HYD' AT ','

INTO v_id v_name v_street,v_city:

Result: v_id - E001

v_name - Emax Technologies.

v_street - Ameerpet

v_city - HYD.

Note: v_id,v_name,v_street,v_city should be declared as Strings.

SHIFT: By Default Shifts to LEFT By 1 Place.

1. SHIFT <Str> <LEFT/RIGHT/CIRCULAR> BY <N> PLACES. (N > 0)

2. SHIFT <Str> LEFT DELETING LEADING <Char>.

3. SHIFT <Str> RIGHT DELETING TRAILING <Char>.

Ex: SHIFT 'Emax'. -> max (Since Default LEFT).

SHIFT 'Emax' CIRCULAR -> maxE

SHIFT 'Emax' RIGHT BY 2 PLACES

-> ' Emax'(First 2 characters are Spaces as it is shifted to 2 Places right.)

SHIFT '000001000' LEFT DELETING LEADING '0'.

Result: 1000. (Deletes all the Left Leading Zeros).

SEARCH : Search for the required Sub String in the Main String.

1. SEARCH <Str> FOR <Str1>.

'str' : a character string (trailing spaces are ignored)

'str.' : any character string between the periods

'*str' - a word ending with "str", including "str"

'str*' - a word beginning with "str", including "str"

RESULT : SY-SUBRC = 0, When Search is Successful.
 = 4, When Un Successful(Not Found)
 SY-FDPOS contains the offset(Position) of the found string .

Comparing strings:

These Special Comparisons can be applied to Strings with types C, D, N, and T.

Operator	Meaning
CO	Contains Only
CN	Contains Not only
CA	Contains Any
NA	contains Not Any
CS	Contains String
NS	contains No String
CP	Matches pattern
NP	Does not match pattern

CO (Contains Only) : IF <Str1> CO <Str2> .

is **TRUE** if <Str1> contains only characters from <Str2>. The comparison is **case-sensitive**. **Trailing blanks are included**.

If the comparison is true, the system field SY-FDPOS contains the length of <Str1>. If it is false, SY-FDPOS contains the offset of the first character of <Str1> that does not occur in <Str2>.

CN (Contains Not only) : IF <Str1> CN <Str2>

is **TRUE** if <Str1> does also contains characters other than those in <Str2>. **The comparison is case-sensitive**. Trailing blanks are included. If the comparison is true, the system field SY-FDPOS contains the offset of the first character of <Str1> that does not also occur in <Str2>. If it is false, SY-FDPOS contains the length of <Str1>.

CA (Contains Any) : <Str1> CA <Str2>

is **TRUE** if <Str1> contains at least one character from <Str2>. The comparison is case-sensitive. If the comparison is true, the system field SY-FDPOS contains the offset of the first character of <Str1> that also occurs in <Str2>. If it is false, SY-FDPOS contains the length of <Str1>.

NA (contains Not Any) : IF <Str1> NA <Str2>

is **TRUE** if <Str1> does not contain any character from <Str2>. The comparison is case-sensitive. If the comparison is true, the system field SY-FDPOS contains the length of <Str1>. If it is false, SY-FDPOS contains the offset of the first character of <Str1> that occurs in <Str2>.

CS (Contains String) : IF <Str1> CS <Str2>

is **TRUE** IF <Str1> contains the String <Str2>. Trailing spaces are ignored and the comparison is **NOT** case-sensitive. If the comparison is true, the system field SY-FDPOS contains the offset of <Str2> in <Str1>. If it is false, SY-FDPOS contains the length of <Str1>.

NS (contains No String) : IF <Str1> NS <Str2>

is true if <Str1> does not contain the String <Str2>. Trailing spaces are ignored and the comparison is **not** case-sensitive. If the comparison is true, the system field SY-FDPOS contains the length of <Str1>. If it is false, SY-FDPOS contains the offset of <Str2> in <Str1>.

CP (Contains Pattern) : IF <Str1> CP <Str2>

is **TRUE** IF <Str1> matches the pattern <Str2>. If <Str2> is of type C, you can use the following wildcards in <Str2>:

- for any character **string**: *
- for any single character: +

Trailing spaces are ignored and the comparison is **not** case-sensitive. If the comparison is true, the system field SY-FDPOS contains the offset of <Str2> in <Str1>. If it is false, SY-FDPOS contains the length of <Str1>.

If you want to perform a comparison on a particular character in <Str2>, place the escape character // in front of it. You can use the escape character // to specify

- characters in upper and lower case
- the wildcard character "*" (enter://*)

- the wildcard character "+" (enter: #+)
- the escape symbol itself (enter: ##)
- blanks at the end of a **String** (enter: # ___)

NP (contains No Pattern) : IF <Str1> NP <Str2>

is **TRUE** if <Str1> does not match the pattern <Str2>. In <Str2>, you can use the same wildcards and escape character as for the **operator** CP.

Trailing spaces are ignored and the comparison is **not** case-sensitive. If the comparison is true, the system field SY-FDPOS contains the length of <Str1>. If it is false, SY-FDPOS contains the offset of <Str2> in <Str1> .

The following table shows the results of executing this program, depending on which **operators** and values of Str1 and Str2.

EXAMPLES :

<Str1>	<operator>	<Str2>	Result	SY-FDPOS
'BD '	CO	'ABCD '	true	5
'BD '	CO	'ABCDE'	false	2
'ABC12'	CN	'ABCD '	true	3
'ABABC'	CN	'ABCD '	false	5
'ABcde'	CA	'Bd '	true	1
'ABcde'	CA	'bD '	false	5
'ABAB '	NA	'AB '	false	0
'ababa'	NA	'AB '	true	5
'ABcde'	CS	'bC '	true	1
'ABcde'	CS	'ce '	false	5
'ABcde'	NS	'bC '	false	1
'ABcde'	NS	'ce '	true	5
'ABcde'	CP	'*b*'	true	1
'ABcde'	CP	'*#b*'	false	5
'ABcde'	NP	'*b*'	false	1
'ABcde'	NP	'*#b*'	true	5

Exercise 5

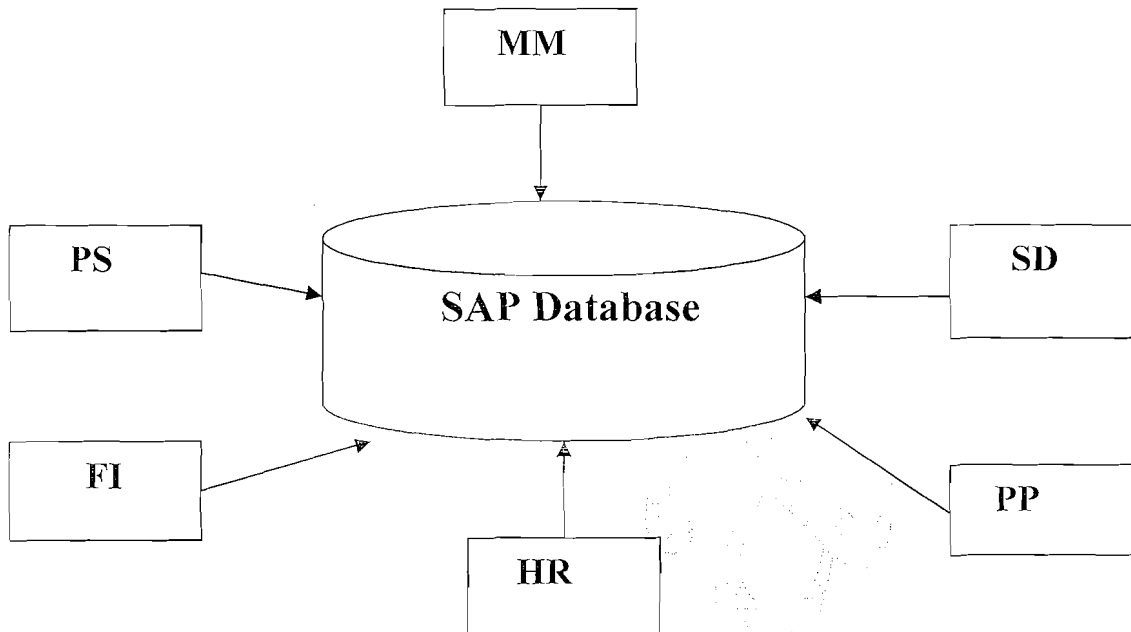
1. Accept a sting and determine its length.
2. Accept a string and number. Write the string that many number of times.
3. Accept two stings and swap their contents.
4. Accept two strings and concatenate into one string.
5. Accept one string with delimiter/ separator with . or :) and split it into two sub strings.
6. Accept a string 'EMAX TECH' and use shift <string> left, < shift> right, shift <string> up to 'TECH'.
7. Accept a string like 'IMAX TECH' , Change first occurrence of 'I' to 'E'. (Use REPLACE command)
8. Accept a string like EMAX TECHNOLOGIES, Change all the occurrence of 'E' to 'G'.
(use TRANSLATE command)
9. Accept two strings and compare the two strings using CO,CA,CS,CP. (output shall be 'true' or 'false' for each comparison).
10. Accept a string 'EMAX TECH', display the output only as 'TECH' using OFFSET command.
11. Accept a string, check whether it is palindrome or not.

6. Internal Tables

Duration in Days - 3(* 2 Hrs)

- a. Introduction & purpose
- b. Declaring Internal Tables
- c. Read Data from Database into ITAB
- d. Processing Data from internal Tables
 - I. Display Data from ITAB
 - II. Adding records (APPEND/INSERT)
 - iii. Modify Records from ITAB (MODIFY)
 - iv. Reading Data from ITAB(READ)
 - v. Delete Records from ITAB (DELETE)
 - vi. Delete Adjacent Duplicate Records
 - vii. Append of Internal Tables Lines (COLLECT)
 - viii. Types of Internal Tables
 - ix. Types of Declaring Internal Tables
 - x. Declaring STANDARD Tables

Note : The Same SAP Database is Accessed by all the departments in the Organization.

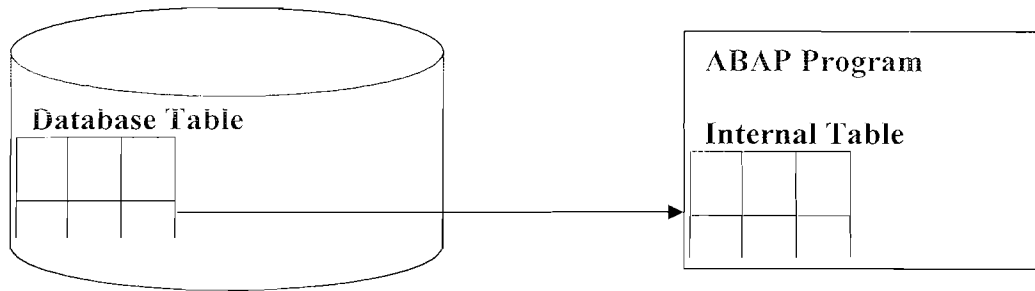


Note : Most Of the times, Each Department want to see the same data in a different view and want to Manipulate according their requirements and that Manipulated data is NOT required by the other departments So that Data in Database(Tables) Should not be changed , When the Changes are required by all the Departments. Instead the Copy of the required Database Tables should be fetched into Program and Manipulate it accordingly, Which doesn't change the Database.

Note: To Maintain the Copy Of the Database Table Data in the Program, the Program should have a variable (Temporary Table) , Which is Nothing but Internal Table.

So that, Internal Table is a Temporary Table (Variable) to Process the Data from Database Table(s).

Database Table & Internal Table :



Features Of Internal Tables:

The data is stored line (record) by line(record) in memory, and each line has the same structure. So that Accessing Data from Internal Tables also record by record.

In ABAP, internal tables are Array (Group) Of Structures, Where the structure is group of fields (record).

A particularly important use for internal tables is for storing and formatting data from a database table within a program and Manipulate according to the user requirements.

Memory Allocation for Internal Tables Is Dynamic Thus internal tables are dynamic data objects, since they can contain any number of lines of **a particular type**.

The only restrictions on the number of lines an internal table may contain are the limits of your system installation. The maximum memory that can be occupied by an internal table (including its internal administration) is 2 gigabytes. A more realistic figure is up to 500 megabytes. An additional restriction for hashed tables is that they may not contain more than 2 million entries.

The line types of internal tables can be any ABAP data types - elementary, structured, or internal tables.

Steps to Declare Internal Table:

- 1) **Declare the Structure of Internal Table, i.e. a User Defined Data type with the required fields, According to the Fields required from the Corresponding Database Table(s).**

```

TYPES : BEGIN OF <TY>,
          F1 TYPE <Ref. Datatype>  ,
          F2 TYPE <Ref. Datatype>  ,
          F3 TYPE <Ref. Datatype>  ,
          ...
        END OF <TY>.

```

Note : TYPES is to define User Defined Datatype.

- 2) **Declare Internal Table i.e. Array of above Structure.**

```
DATA <Itab> TYPE TABLE OF <TY>.
```

Example: Declare an ITAB to Maintain Data From Table T001(Company Code Data).

Note : While Declaring <ITAB>, refer the Corresponding Database Table i.e T001 Here.

*DECLARE the Required Datatype

```

TYPES : BEGIN OF TY_T001,
          BUKRS TYPE BUKRS, "Company Code
          BUTXT TYPE BUTXT, "Company Name
          ORT01 TYPE ORT01, "City
          LAND1 TYPE LAND1, "Country Key
        END OF TY_T001.

```

*Declare Internal Table From the above Structure

```
DATA IT_T001 TYPE TABLE OF TY_T001.
```

Syntax to Select the Data From Database Table into Internal Table :

```

SELECT <F1>
      <F2>
      <F3>
      ...
INTO TABLE <ITAB>
FROM <DBT>
WHERE <Condition If Any> .

```

Note : Make Sure that the structure(Order) of Fields in the SELECT and Internal Table Should be Same , Because the Content Of 1st Field in the SELECT is transferred to the 1st Field Of <ITAB> and Similarly 2nd,3rd, etc.

Example Program : Display the List Of German Companies` s Details
(Company Code,Name,City,Country) From T001.

***DECLARE the Required Datatype**

```

TYPES : BEGIN OF TY_T001,
          BUKRS TYPE BUKRS, "Company Code
          BUTXT TYPE BUTXT, "Company Name
          ORT01 TYPE ORT01, "City
          LAND1 TYPE LAND1, "Country Key
        END   OF TY_T001.

```

***Declare Internal Table From the above Structure**

```

DATA IT_T001 TYPE TABLE OF TY_T001.

```

***Fetch Data From T001 into IT_T001.**

```

SELECT BUKRS
       BUTXT
       ORT01
       LAND1
INTO TABLE IT_T001
FROM T001
WHERE LAND1 = 'DE'. "Germany

```

Note : After the SELECT Data From Database Table is transferred to Internal Table .

***Displaying Data From Internal Table**

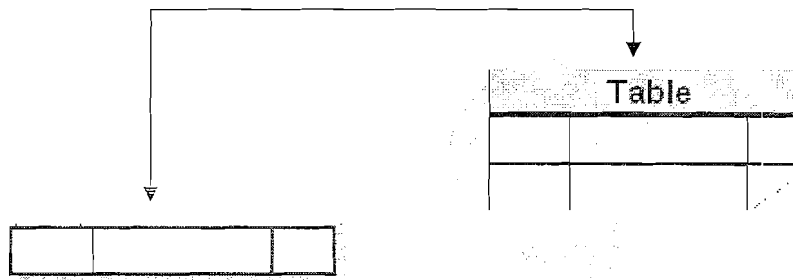
*Since Data in Internal Table is Stored record by record, Accessing/Displaying

*Data From Internal Table is also record by record. So that There is a Special Loop

*Which Starts with the 1st Record and Ends with Last Record Of <ITab> By Default.

*Keep the Internal Table in a LOOP, LOOP Points to the First record by Default
*and Collect it into <WA> and Process(Print) the Data from the <WA>.

LOOP AT <ITAB> INTO <WA>.



Work area

ENDLOOP.

PROGRAM :

```
*&-----
*& Report      ZGDEMO_DISPLAY_COMPANIES_DE
* PURPOSE     : TO DISPLAY LIST OF GERMAN COMPANIES AND THEIR DETAILS
* AUTHOR      : GANAPATI ADIMULAM
* COMPANY     : EMAX TECHNOLOGIES
*&-----
```

REPORT ZGDEMO_DISPLAY_COMPANIES_DE .

*DECLARE the Required Datatype

```
TYPES : BEGIN OF TY_T001,
        BUKRS TYPE BUKRS,   "Company Code
        BUTXT TYPE BUTXT,   "Company Name
        ORT01 TYPE ORT01,   "City
        LAND1 TYPE LAND1,   "Country Key
        END OF TY_T001.
```

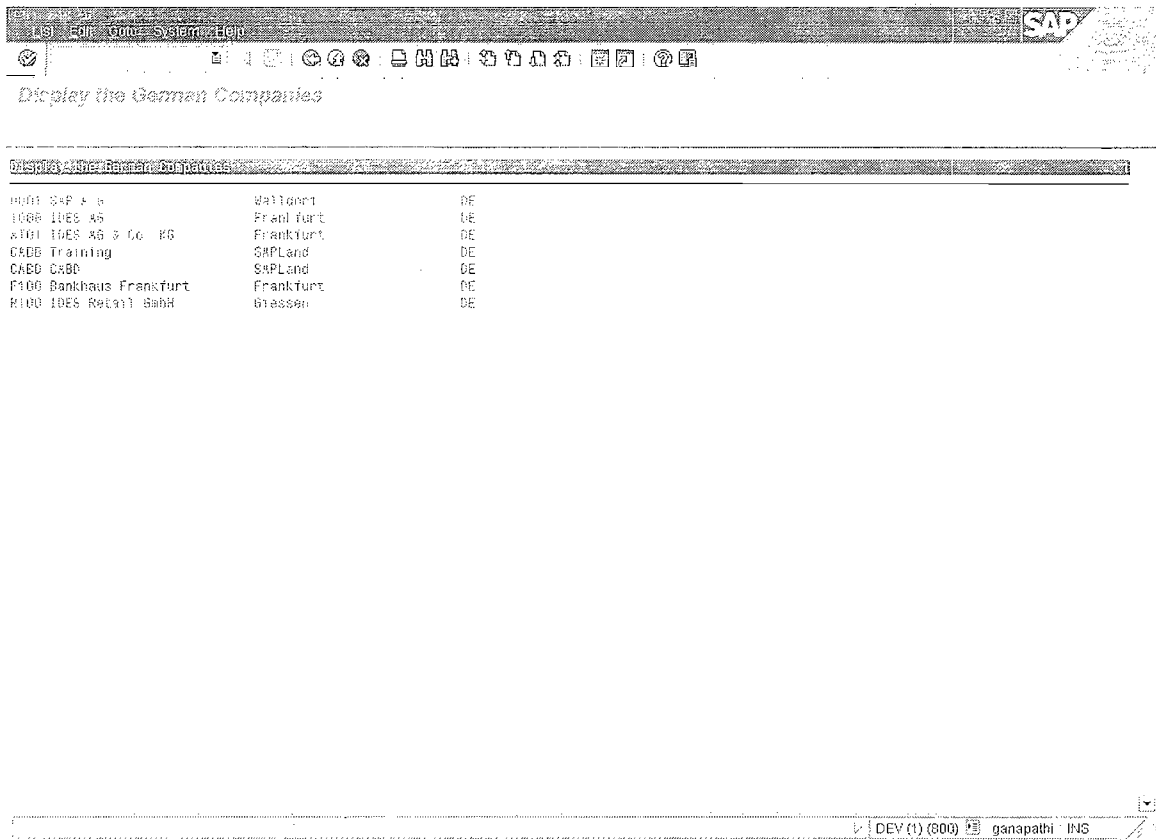
*Declare Internal Table From the above Structure

```
DATA IT_T001 TYPE TABLE OF TY_T001.
DATA WA_T001 TYPE TY_T001.
```

```
etch Data From T001 into IT_T001
SELECT BUKRS
      BUTXT
      ORT01
      LAND1
INTO TABLE IT_T001
FROM T001
WHERE LAND1 = 'DE'.
```

```
*Display Data From IT_T001
LOOP AT IT_T001 INTO WA_T001.
  WRITE : / WA_T001-BUKRS,
          WA_T001-BUTXT,
          WA_T001-ORT01,
          WA_T001-LAND1.
ENDLOOP.
```

OUTPUT : EXECUTE THE PROGRAM



OPERATIONS ON INTERNAL TABLES :

DAY-2

COPYING (Adding Multiple Records):

A) Copy at the END Of Internal Table

APPEND LINES OF <ITAB1> FROM <N1> TO <N2> TO <ITAB2>.

B) Copy From the given Location

INSERT LINES OF <ITAB1> FROM <N1> TO <N2> TO <ITAB2>
INDEX <N>.

NOTE: INSERT With NO INDEX Acts as APPEND Only.

FROM <N1> TO <N2> is Optional, If We Ignore it, All the Records are Transferred, Else Only the records from <N1> TO <N2> are Transferred.

ADDING SINGLE RECORD :

A) ADD at the END Of Internal Table

APPEND <WA> TO <ITAB>.

B) ADD at the given Location

INSERT <WA> INTO <ITAB> INDEX <N>. (N > 0)

NOTE: INSERT With NO INDEX Acts as APPEND Only.

NOTE: Fill the Data into <WA> and Transfer to <ITAB>.

Finding No Of Records :

DESCRIBE TABLE <ITAB> LINES <V_LINES>. (V_LINES TYPE I).

The No Of Records from <ITAB> is Collected Into V_LINES.

SORTing : To Arrange the Records into Ascending / Descending Groups .

SORT <ITAB> ASCENDING / DESCENDING BY <F1> <F2>...

NOTE : Sorting is Ascending by Default.

NOTE :The default key is made up of the Non-Numeric fields of the table line in the order in which they occur.

Accessing/Reading Single Record :

READ TABLE <ITAB> INTO <WA> INDEX <N>.

OR

READ TABLE <ITAB> INTO <WA> WITH KEY <Condition> BINARY

SEARCH.

NOTE: Make Sure that the Internal Table is Sorted to Apply BINARY SEARCH.

Accessing Multiple Records :

LOOP AT <ITAB> INTO <WA> FROM <N1> TO <N2>.

***Process the Data From <WA>.**

ENDLOOP.

OR

LOOP AT <ITAB> INTO <WA> WHERE <Condition>

***Process the Data From <WA>.**

ENDLOOP.

DELETING Records :

Single Record : DELETE <ITAB> INDEX <N>.

Multiple Records : DELETE <ITAB> WHERE <Condition>.

DELETE <ITAB> FROM <N1> TO <N2>.

DELETING ADJACENT DUPLICATES :

NOTE: Make Sure that the Duplicates Should be Adjacent. Which Can be Done through SORTing.

So that , Sorting the <ITAB> is Mandatory.

NOTE : The Duplication Of Record(s), Depends On the Comparing Fields.

DELETE ADJACENT DUPLICATES FROM <ITAB>

COMPARING <F1> <F2>

Note : Make Sure that , the <ITAB> is Sorted by all the Comparing Fields in the Same Order..

MODIFY :

Either Single / Multiple Records Modification is always through <WA>.

***Fill the New Data into <WA> Fields.**

MODIFY <ITAB> FROM <WA> TRANSPORTING <F1> <F2> ...

WHERE <Condition>.

NOTE : The Where Condition , Decides the No Of Records to be Modified.

EXAMPLE PROGRAM WITH ALL THE INTERNAL TABLE OPERATIONS :

```

*&-----
*& Report      ZGDEMO_DISPLAY_COMPANIES_DE
* PURPOSE : TO DISPLAY LIST OF INTERNAL TABLE OPERATIONS
* AUTHOR  : GANAPATI.ADIMULAM
* COMPANY : EMAX TECHNOLOGIES
*&-----

```

```
REPORT ZGDEMO_ITAB_OPERATIONS
```

```

*DECLARE the Required Datatype
TYPES : BEGIN OF TY_T001,
        BUKRS TYPE BUKRS, "Company Code
        BUTXT TYPE BUTXT, "Company Name
        ORT01 TYPE ORT01, "City
        LAND1 TYPE LAND1, "Country Key
        END OF TY_T001.

```

```

*Declare Internal Table From the above Structure
DATA IT_T001 TYPE TABLE OF TY_T001.
DATA WA_T001 TYPE TY_T001.

```

```

*Fetch Data From T001 into IT_T001.
SELECT BUKRS
       BUTXT
       ORT01
       LAND1
INTO TABLE IT_T001
FROM T001
WHERE LAND1 = 'DE'.

```

```

*SORT
SORT IT_T001 DESCENDING BY BUKRS.

```

```

WRITE : / 'LIST OF GERMAN COMPANIES IN DESCENDING ORDER'
COLOR 1.
ULINE.

```

```

*Display Data From IT_T001
LOOP AT IT_T001 INTO WA_T001.
WRITE : / SY-TABIX,
        WA_T001-BUKRS,

```

```
        WA_T001-BUTXT,  
        WA_T001-ORT01,  
        WA_T001-LAND1.  
ENDLOOP.  
  
*ADD SOME MORE RECORDS  
CLEAR WA_T001. "CLEARs THE CONTENTS  
  
        WA_T001-BUKRS = 'E001'.  
        WA_T001-BUTXT = 'EMAX TECHNOLOGIES'.  
        WA_T001-ORT01 = 'HYDERABAD'.  
APPEND WA_T001 TO IT_T001.  
  
*INSERT  
CLEAR WA_T001. "CLEARs THE CONTENTS  
  
        WA_T001-BUKRS = 'I001'.  
        WA_T001-BUTXT = 'IMAX TECHNOLOGIES'.  
        WA_T001-ORT01 = 'HYDERABAD'.  
INSERT WA_T001 INTO IT_T001 INDEX 1.  
  
SKIP 2.  
WRITE : / 'DATA AFTER APPEND AND INSERT' COLOR 1.  
ULINE.  
  
*Display Data From IT_T001  
LOOP AT IT_T001 INTO WA_T001.  
    WRITE : / SY-TABIX,  
            WA_T001-BUKRS,  
            WA_T001-BUTXT,  
            WA_T001-ORT01,  
            WA_T001-LAND1.  
  
ENDLOOP.  
  
*DELETE  
DELETE IT_T001 FROM 4 TO 5.  
SKIP 2.  
WRITE : / 'DATA AFTER DELETE' COLOR 1.  
ULINE.  
  
*Display Data From IT_T001  
LOOP AT IT_T001 INTO WA_T001.  
    WRITE : / SY-TABIX,  
            WA_T001-BUKRS,  
            WA_T001-BUTXT,
```

```
        WA_T001-ORT01,  
        WA_T001-LAND1.  
ENDLOOP.  
  
*MODIFY  
*SET COUNTRY KEY TO 'IN' FOR BOTH COMPANY CODEs 'E001' AND  
'I001'.  
CLEAR WA_T001.  
    WA_T001-LAND1 = 'IN'.  
    MODIFY IT_T001 FROM WA_T001 TRANSPORTING LAND1  
        WHERE BUKRS = 'E001' OR  
            BUKRS = 'I001' .  
  
IF SY-SUBRC = 0 . "SUCCESSFULLY MODIFIED"  
SKIP 2.  
WRITE : / 'DATA AFTER MODIFY' COLOR 1.  
ULINE.  
  
*Display Data From IT_T001  
LOOP AT IT_T001 INTO WA_T001.  
    WRITE : / SY-TABIX,  
        WA_T001-BUKRS,  
        WA_T001-BUTXT,  
        WA_T001-ORT01,  
        WA_T001-LAND1.  
  
ENDLOOP.  
  
ENDIF.  
  
*READ  
SORT IT_T001 BY BUKRS ASCENDING.  
CLEAR WA_T001.  
READ TABLE IT_T001 INTO WA_T001 WITH KEY BUKRS = '0001'  
BINARY SEARCH.  
SKIP 2.  
WRITE : / 'DATA AFTER READ' COLOR 1.  
ULINE.  
  
*Display Data From IT_T001  
    WRITE : / SY-TABIX,  
        WA_T001-BUKRS,  
        WA_T001-BUTXT,  
        WA_T001-ORT01,  
        WA_T001-LAND1.
```

```
*DELETE ADJACENT DUPLICATES
*SINCE WE DON'T HAVE DUPLICATES ON BUKRS, WE PREPARE
DUPLICATES
DO 3 TIMES.
  APPEND LINES OF IT_T001 TO IT_T001.
ENDDO.

SORT IT_T001 BY BUKRS ASCENDING.

SKIP 2.
WRITE : / 'DATA AFTER ADDING DUPLICATE RECORDS' COLOR 1.
ULINE.

LOOP AT IT_T001 INTO WA_T001.
*Display Data From IT_T001
  WRITE : / SY-TABIX,
          WA_T001-BUKRS,
          WA_T001-BUTXT,
          WA_T001-ORT01,
          WA_T001-LAND1.
ENDLOOP.

*DELETE ADJACENT DUPLICATES
DELETE ADJACENT DUPLICATES FROM IT_T001 COMPARING BUKRS.

SKIP 2.
WRITE : / 'DATA AFTER ADJACENT DUPLICATES' COLOR 1.
ULINE.

LOOP AT IT_T001 INTO WA_T001.
*Display Data From IT_T001
  WRITE : / SY-TABIX,
          WA_T001-BUKRS,
          WA_T001-BUTXT,
          WA_T001-ORT01,
          WA_T001-LAND1.
ENDLOOP.
```

OUTPUT OF THE PROGRAM :**EXECUTE THE PROGRAM**

Display the German Companies

Display the German Companies

LIST OF GERMAN COMPANIES IN DESCENDING ORDER

1	8100 IDES Retail GmbH	Blessen	DE
2	F100 Banhaus Frankfurt	Frankfurt	DE
3	G880 CRB	SMPLand	DE
4	G880 Training	SMPLand	DE
5	AT01 IDES AG & Co. KG	Frankfurt	DE
6	1000 IDES AG	Frankfurt	DE
7	0001 SAP A.G.	Walldorf	DE

DATA AFTER APPEND AND INSERT

1	1001 INAX TECHNOLOGIES	HYDERABAD	
2	8100 IDES Retail GmbH	Blessen	DE
3	F100 Banhaus Frankfurt	Frankfurt	DE
4	G880 CRB	SMPLand	DE
5	G880 Training	SMPLand	DE
6	AT01 IDES AG & Co. KG	Frankfurt	DE
7	1000 IDES AG	Frankfurt	DE
8	0001 SAP A.G.	Walldorf	DE
9	E001 ERAT TECHNOLOGIES	HYDERABAD	

DATA AFTER DELETE

1	1001 INAX TECHNOLOGIES	HYDERABAD	
2	8100 IDES Retail GmbH	Blessen	DE
3	F100 Banhaus Frankfurt	Frankfurt	DE
4	AT01 IDES AG & Co. KG	Frankfurt	DE
5	1000 IDES AG	Frankfurt	DE
6	0001 SAP A.G.	Walldorf	DE
7	E001 ERAT TECHNOLOGIES	HYDERABAD	

DEV (1) (800) ganapati INS

Internal Tables

We Never Compromise in Quality, Would You?

Display the German Companies

Display the German Companies

DATA AFTER MODIFY

1	1001	EMAX TECHNOLOGIES	HYDERABAD	IN
2	R100	IDES Retail GmbH	Gießen	DE
3	F100	Bankhaus Frankfurt	Frankfurt	DE
4	AT01	IDES AS & Co. KG	Frankfurt	DE
5	1000	IDES AG	Frankfurt	DE
6	0001	SAP A.G.	Walldorf	DE
7	0001	EMAX TECHNOLOGIES	HYDERABAD	IN

DATA AFTER READ

1	0001	SAP A.G.	Walldorf	DE
---	------	----------	----------	----

DATA AFTER ADDING DUPLICATE RECORDS

1	0001	SAP A.G.	Walldorf	DE
2	0001	SAP A.G.	Walldorf	DE
3	0001	SAP A.G.	Walldorf	DE
4	0001	SAP A.G.	Walldorf	DE
5	0001	SAP A.G.	Walldorf	DE
6	0001	SAP A.G.	Walldorf	DE
7	0001	SAP A.G.	Walldorf	DE
8	0001	SAP A.G.	Walldorf	DE
9	1000	IDES AG	Frankfurt	DE
10	1000	IDES AG	Frankfurt	DE
11	1000	IDES AG	Frankfurt	DE
12	1000	IDES AG	Frankfurt	DE
13	1000	IDES AG	Frankfurt	DE
14	1000	IDES AG	Frankfurt	DE
15	1000	IDES AG	Frankfurt	DE
16	1000	IDES AG	Frankfurt	DE
17	AT01	IDES AS & Co. KG	Frankfurt	DE

DEV (1) (200) ganapathi INS

Display the German Companies

Display the German Companies

18	AT01	IDES AS & Co. KG	Frankfurt	DE
19	AT01	IDES AS & Co. KG	Frankfurt	DE
20	AT01	IDES AS & Co. KG	Frankfurt	DE
21	AT01	IDES AS & Co. KG	Frankfurt	DE
22	AT01	IDES AS & Co. KG	Frankfurt	DE
23	AT01	IDES AS & Co. KG	Frankfurt	DE
24	AT01	IDES AS & Co. KG	Frankfurt	DE
25	0001	EMAX TECHNOLOGIES	HYDERABAD	IN
26	0001	EMAX TECHNOLOGIES	HYDERABAD	IN
27	0001	EMAX TECHNOLOGIES	HYDERABAD	IN
28	0001	EMAX TECHNOLOGIES	HYDERABAD	IN
29	0001	EMAX TECHNOLOGIES	HYDERABAD	IN
30	0001	EMAX TECHNOLOGIES	HYDERABAD	IN
31	0001	EMAX TECHNOLOGIES	HYDERABAD	IN
32	0001	EMAX TECHNOLOGIES	HYDERABAD	IN
33	F100	Bankhaus Frankfurt	Frankfurt	DE
34	F100	Bankhaus Frankfurt	Frankfurt	DE
35	F100	Bankhaus Frankfurt	Frankfurt	DE
36	F100	Bankhaus Frankfurt	Frankfurt	DE
37	F100	Bankhaus Frankfurt	Frankfurt	DE
38	F100	Bankhaus Frankfurt	Frankfurt	DE
39	F100	Bankhaus Frankfurt	Frankfurt	DE
40	F100	Bankhaus Frankfurt	Frankfurt	DE
41	1001	EMAX TECHNOLOGIES	HYDERABAD	IN
42	1001	EMAX TECHNOLOGIES	HYDERABAD	IN
43	1001	EMAX TECHNOLOGIES	HYDERABAD	IN
44	1001	EMAX TECHNOLOGIES	HYDERABAD	IN
45	1001	EMAX TECHNOLOGIES	HYDERABAD	IN
46	1001	EMAX TECHNOLOGIES	HYDERABAD	IN
47	1001	EMAX TECHNOLOGIES	HYDERABAD	IN
48	1001	EMAX TECHNOLOGIES	HYDERABAD	IN
49	R100	IDES Retail GmbH	Gießen	DE
50	R100	IDES Retail GmbH	Gießen	DE
51	R100	IDES Retail GmbH	Gießen	DE
52	R100	IDES Retail GmbH	Gießen	DE

DEV (1) (200) ganapathi INS

Display the German Companies

53	R100	IDES Retail GmbH	Gießen	DE
54	R100	IDES Retail GmbH	Gießen	DE
55	R100	IDES Retail GmbH	Gießen	DE
56	R100	IDES Retail GmbH	Gießen	DE

DATA AFTER ADJACENT DUPLICATES

1	990	ERP	Ballhof	DE
2	1000	IDES AG	Frankfurt	DE
3	AT01	IDES AG & Co. KG	Frankfurt	DE
4	F001	EMAX TECHNOLOGIES	HYDERABAD	IN
5	F100	Bankhaus Frankfurt	Frankfurt	DE
6	1001	EMAX TECHNOLOGIES	HYDERABAD	IN
7	R100	IDES Retail GmbH	Gießen	DE

DEV (1) (200) ganapathi INS

MISCELLANEOUS:**DAY-3**

HEADER LINE: Is the Default (Implicit) Work Area, Defined by the System, with the name Of Internal Table.

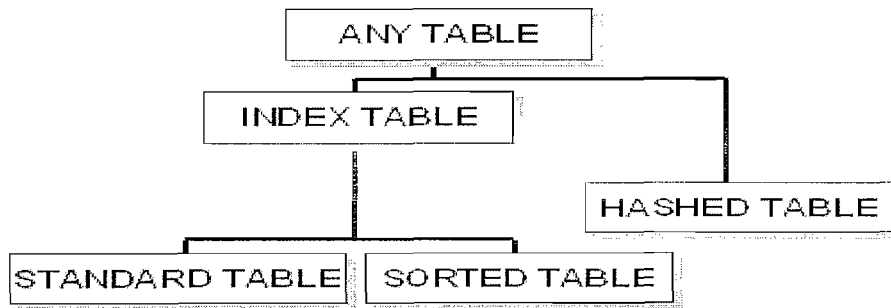
DATA IT_T001 TYPE TABLE OF T001 WITH HEADER LINE.

Note : Here Both Internal Table and Work Areas are Defined with the Same Name i.e IT_T001.

NOTE : <ITAB>s With **HEADER LINE** are Not Recommended , Instead SAP Recommends Explicit Work Areas.

TYPEs Of INTERNAL TABLEs :

Hierarchy of Table Types

STANDARD TABLEs :

- . Are the Default Internal Table
- . The key of a standard table is always non-unique So Duplicates are Allowed.
- . DATA <ITAB> TYPE STANDARD TABLE OF <TY>.
- . Records Can be accessed through both INDEX and Condition.
 - READ TABLE <ITAB> INTO <WA> INDEX <N>.
 - OR
 - READ TABLE <ITAB> INTO <WA> WITH KEY <Condition>.
- . <ITAB> Can be Sorted
- . Accessing/Searching time for the record depends on the No Of Records Because Searching is either Liner or Binary.

SORTED TABLES

Note: Records are always in Sorted Order.

This is the most appropriate type if you need a table which is sorted as you fill it. You fill sorted tables using the INSERT statement. Entries are inserted according to the sort sequence defined through the table key.

The response time for key access is logarithmically proportional to the number of table entries, since the system always uses a binary search.

Note : Records Can be accessed through Both INDEX and with Key(Condition).

Note: Sorted Internal Tables Cannot be Sorted again.

Sorted Internal Tables are always, either UNIQUE / NON UNIQUE.
I.e. Sorted Internal Tables Cannot be Declared without UNIQUE/NON-UNIQUE Keywords.

```
DATA <ITAB> TYPE SORTED TABLE OF <TY> WITH UNIQUE/NON-UNIQUE KEY <F1> <F2> ..
```

Hashed tables :

This is the most appropriate type for any table where the main operation is key access.

Like database tables, hashed tables always have a **unique key**.

```
DATA <ITAB> TYPE HASHED TABLE OF <TY> WITH UNIQUE/NON-UNIQUE KEY <F1> <F2> ..
```

You cannot access a hashed table using its **INDEX**.

The response (Search) time doesn't depend on the no of records, Instead it is always access remains constant, regardless of the number of table entries.

Hashed tables are useful if you want to construct and use an internal table which **resembles a database table** or for processing large amounts of data.

Special Features of Standard Tables:

Sorted tables, Hashed tables are only introduced in **Release 4.0**, standard tables already existed several releases previously.

Defining a line type, table type, and tables **without a header line have only been possible** since **Release 3.0**. For this reason, there are certain features of standard tables that still exist for compatibility reasons.

Standard Tables Before Release 3.0

Before Release 3.0, internal tables all had header lines and a flat-structured line type. **There were no independent table types**. You could only create a table object using the **OCCURS** addition in the DATA statement, followed by a declaration of a flat structure:

```
DATA: BEGIN OF <ITAB> OCCURS <n>,  
      <f1>  
      <f2>  
      ....  
      <fn>  
END OF <itab>.
```

This statement declared an internal table <itab> with the line type defined following the OCCURS addition. Furthermore, all internal tables had header lines.

The number <n> in the OCCURS addition had the same meaning as in the INITIAL SIZE addition from Release 4.0. Entering '0' had the same effect as omitting the INITIAL SIZE addition. In this case, the initial size of the table is determined by the system.

Standard Tables From Release 3.0

Since Release 3.0, it has been possible to create table types using

TYPES <t> TYPE|LIKE <linetype> OCCURS <n>.

and table objects using

DATA <itab> TYPE|LIKE <linetype> OCCURS <n> [WITH HEADER LINE].

The effect of the OCCURS addition is to construct a standard table with the data type <linetype>. The line type can be any data type.

The above statements are still possible in Release 4.0, and have the same function as the following statements:

TYPES <itab> TYPE|LIKE [STANDARD] TABLE OF <linetype>.

NOTE : OCCURS <0> : Allocates the Initial Memory 8KB and the system keep on allocates by 8KB , whenever it is required.

OCCURS <N> : Allocates Memory For <N> records initially and Keep on allocates for <N> records , whenever it requires.

INITIALIZING INTERNAL TABLES:

NOTE: Whenever the same Internal Table is used again and again in the same Program, it is better to INITIALIZE (Clear the Current Current) and before we use it again.

CLEAR:

CLEAR <Variable>.

NOTE: The <Variable> Can be a normal Variable, Work Area, Internal Table etc.

So that CLEAR Can Clear the Contents Of the Corresponding Variable.

CLEAR <WA>. - Clears Work Area

CLEAR <ITAB>. - Clears Internal Table

Note: Clearing the Internal Table means, to take to the state that it was in immediately after it is declared it.

Note: If you are using internal tables with header lines, remember that the header line and the body of the table have the **same name**. If you want to address the body of the table in a comparison, you must place two brackets ([]) after the table name.

CLEAR <itab>[]. “Clears Internal Table

CLEAR <Itab>. “Clears the Work Area

REFRESH:

Always Works For Internal Table Only.

This always applies to the body of the table.

As with the CLEAR statement, the memory used by the table before you initialized it remains allocated. To release the memory space, use the statement **FREE**.

FREE: Always Works For Internal Table Only.

You can use FREE to initialize an internal table and also to release its memory space

That is allocated and which is not Possible through CLEAR and REFRESH.

Note : Both CLEAR and REFRESH Can Clear Only the Contents but they cannot release the Memory Occupied. After a FREE statement, you can address the internal table again. When you refill the table, the system has to allocate new memory space to the lines.

EXAMPLE PROGRAM ON INITIALIZING INTERNAL TABLES:

```

*****
* PROGRAM       :      ZDEMO_INITIALIZE_ITABS
* AUTHOR        :      GANAPATI .ADIMULAM
* START DATE    :      30/06/2006
* PURPOSE       :      IS TO WORK WITH INITIALIZING ITABS
* COPIED FROM   :      NA
*****
* MODIFICATION LOG:
*CHANGE REQUEST :      C11EMAX4756
*-----
*MOD-001       :      DETAILS OF MODIFICATION 001
*MOD-002       :      DETAILS OF MODIFICATION 002
*SUPPLIERS     :      EMAX TECHNOLOGIES
*****

DATA: BEGIN OF TY_ T001,
      BUKRS TYPE BUKRS,
      BUTXT TYPE BUTXT,
      END OF TY_ T001.

DATA: IT_ T001  TYPE TABLE OF TY_ T001 WITH HEADER LINE,
      ITI_ T001 TYPE TABLE OF TY_ T001,
      WA_ T001 TYPE TY_ T001.

```

```
*APPEND 1ST RECORD
WA_T001-BUKRS = '1000'.
WA_T001-BUTXT = 'EMAX TECHNOLOGIES'.
APPEND WA_T001 TO IT_T001.
*APPEND 2ND RECORD
WA_T001-BUKRS = '2000'.
WA_T001-BUTXT = 'CLARION PARK IT'.
APPEND WA_T001 TO IT_T001.
REFRESH IT_T001.
IF IT_T001 IS INITIAL.
WRITE 'ITAB IS EMPTY'.
FREE IT_T001.
ENDIF.
```

OUTPUT:

ITAB is empty.

Appending Summarized Lines

The following statement allows you to summate entries in an internal table:

COLLECT <wa> **INTO** <itab>.

<itab> must have a flat line type, and all of the fields that are not part of the table key must have a **numeric type** (F, I, or P). You specify the line that you want to add in a work area that is compatible with the line type.

When the line is inserted, the system checks whether there is already a table entry that matches the key. If there is no corresponding entry already in the table, the COLLECT statement has the same effect as inserting the new line. If an entry with the same key already exists, the COLLECT statement does not append a new line, but adds the contents of the numeric fields in the work area to the contents of the numeric fields in the existing entry.

You should only use the COLLECT statement if you want to create summarized tables. If you use other statements to insert table entries, you may end up with duplicate entries.

SYNTAX FOR INTERNAL TABLE OPERATIONS**(WITH & WITHOUT HEADER LINES)**

The following table shows the statements that you must use for internal tables without a header line, and the equivalent statements that you can use for internal tables with a header line:

Operations without header line	Operations with header line
Operations for all Table Types	
INSERT <wa> INTO TABLE <itab>.	INSERT TABLE ITAB.
COLLECT <wa> INTO <itab>.	COLLECT <itab>.
READ TABLE <itab> ... INTO <wa>.	READ TABLE <itab> ...
MODIFY TABLE <itab> FROM <wa> ...	MODIFY TABLE <itab> ...
MODIFY <itab> FROM <wa> ...WHERE ...	MODIFY <itab> ... WHERE ...
DELETE TABLE <itab> FROM <wa>.	DELETE TABLE <itab>.
LOOP AT ITAB INTO <wa> ...	LOOP AT ITAB ...
Operations for Index Tables	
APPEND <wa> TO <itab>.	APPEND <itab>.
INSERT <wa> INTO <itab> ...	INSERT <itab> ...
MODIFY <itab> FROM <wa> ...	MODIFY <itab> ...

Using the header line as a work area means that you can use shorter statements; however, they are not necessarily easier to understand, since you cannot immediately recognize the origin and target of the assignment. Furthermore, the fact that the table and its header line have the same name can cause confusion in operations with entire internal tables. To avoid confusion, you should use internal tables with differently-named work areas.

Exercise

1. Declare an internal table for the below fields from CSKS and display the data :
Controlling Area, Cost Centre, Valid-from date, Valid-to date, Department, Person Responsible.
2. Declare an internal table with fields *Profit Centre, Controlling Area, Valid-from date, Valid-to date, Department, Person Responsible.* Display these fields with column headings (Table CEPC – Profit Center Master Data).
3. Declare an internal table with following fields:
Sales Docu.No, Date, Customer No and Sales Docu. Type from table VBAK
Item Details like Item No, Material No, Quantity and Price from VBAP.

Sort the table according to Sales Doc.No and display the contents
4. Declare an internal table it_bsis, having a similar structure as table BSIS, explore all possible methods to create the internal table with header line or without header line
(use data types data begin of , end of, data like , data include structure etc) Populate the internal table with contents of BSIS.Sort the table according to company code and display the contents.
5. Declare an internal table with the following fields and append the records , refer the DDIC table T001 for declaration.

Co.Code,	Co.Name	Street	City	Country
0001	E max Technologies	Ameerpet		In
0002	Clarion Park Technologies	SR Nagar		In
0003	Innate Technologies	Jublee Hills		In

Apply all the below Operations to the Internal Table:

i) Modify all the records with City 'Hyderabad'.

ii) Append the following records to the above internal table.

0003, I max Technologies,Begumpet,Hyderabad,In.

0002, G max Technologies,Madhapur,Hyderabad,In.

Delete the Adjacent Duplicate Records from the above internal table by comparing Co. Codes.

Note: Display the Internal Table after each above operation.

- 6. Explain the Different Types Of Internal tables ?**
- 7. Explain and Write Down the SYNTAX for all the ITAB Operations Like**
 - a. APPEND and INSERT**
 - b. READ**
 - c. MODIFY**
 - d. DELETE**
 - e. DELETE ADJACENT**
 - f. SORT**
 - g. COLLECT**

7. Open SQL

Duration in Days - 2(* 2 Hrs)

- A. Accessing Database in R/3 system
- B. Standard SQL
- C. Database interface
- D. Open SQL
 - i. Types of Command
 - 1. DDL (Data Definition Language)
 - 2. DML (Data Manipulation Language)
 - a. Changing Data
(INSERT, UPDATE
MODIFY, DELETE)
 - b. Reading Data
 - i. SELECT
 - ii. SELECT SINGLE Vs
UP TO 1 ROWS
 - iii. INNER JOIN
 - iv. OUTER JOIN
 - v. FOR ALL ENTRIES
 - 3. DCL (Data Control Language)
 - a. COMMIT WORK
 - b. ROLLBACK WORK

Accessing the Database in the R/3 System

In the R/3 System, long-life data is stored in relational database tables. In a relational database model, the real world is represented by tables. A table is a two-dimensional matrix, consisting of lines and columns (fields).

Standard SQL

SQL (Structured Query Language) is a largely standardized language for accessing relational databases. It can be divided into three areas:

- Data Manipulation Language (DML)

Statements for reading and changing data in database tables.

- Data Definition Language (DDL)

Statements for creating and administering database tables.

- Data Control Language (DCL)

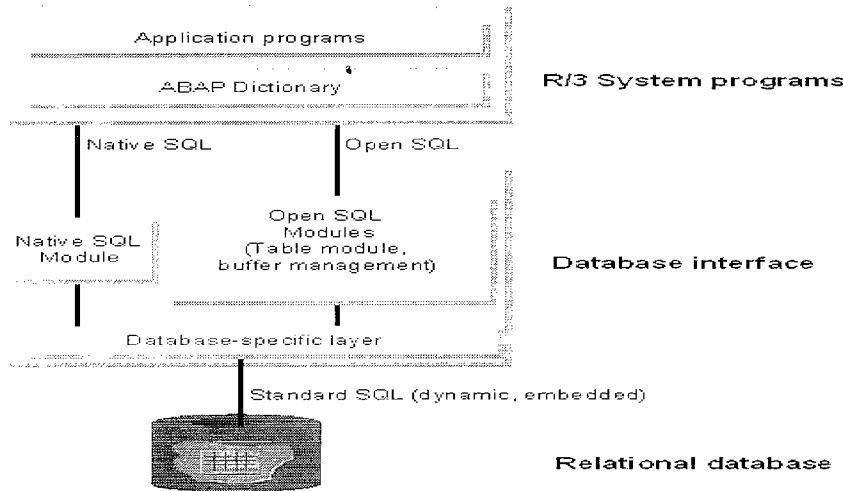
Statements for authorization and consistency checks.

Note : The SQL is Database Dependent.

Each database has a programming interface that allows you to access the database tables using SQL statements. The SQL statements in these programming interfaces are not fully standardized. **To access a specific database system, you must refer to the documentation of that system for a list of the SQL statements available and their correct syntax(Since it is Database Dependent).**

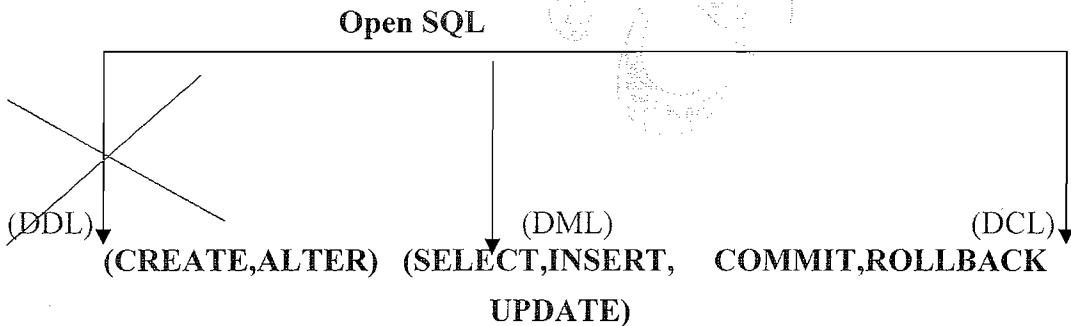
The Database Interface

To make the R/3 System independent of the database system with which you use it despite the differences in the SQL syntax between various databases, each work process on an application server has a database interface. The R/3 System communicates with the database by means of this **interface**. **The database interface converts all of the database requests from the R/3 System into the correct Standard SQL statements for the database system.**



There are two ways of accessing the database from a program - with **Open SQL** or **Native SQL**.

Open SQL, Which is Database Independent and Integrated with ABAP Workbench.



Note: Open SQL Doesn't Support DDL so that we have DATA DICTIONARY to Support the DDL Operations.

Open SQL statements are a subset of Standard SQL that is fully integrated in ABAP. They allow you to access data irrespective of the database system that the R/3 installation is using.

Native SQL:

Native SQL is only loosely integrated into ABAP, and allows access to all of the functions contained in the programming interface of the respective database system. Unlike Open SQL statements, Native SQL statements are not checked and converted, but instead are sent directly to the database system. When you use Native SQL, the function of the database-dependent layer is minimal. Programs that use Native SQL are specific to the database system for which they were written. When writing R/3 applications, you should avoid using Native SQL wherever possible. It is used, however, in some parts of the R/3 Basis System - for

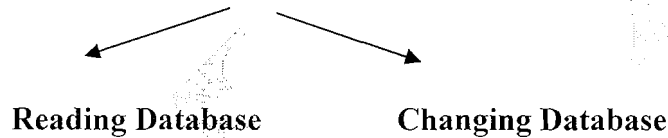
example, for creating or changing table definitions in the ABAP Dictionary.

The ABAP Dictionary uses the DDL part of Open SQL to create and change database tables. Open SQL statements can only access tables that exist in the ABAP Dictionary.

Return Codes

SY-SUBRC	After every Open SQL statement, the system field SY-SUBRC contains the value 0 if the operation was successful, a value other than 0 if not.
SY-DBCNT	After an open SQL statement, the system field SY-DBCNT contains the number of database lines processed.

DML (Data Manipulation Language) Commands:



Note: Reading DATA is through SELECTs, Which will be discussed later.

Working with Database Changing Operations:

A set of statements that allow you to change data in the database.

You can insert, change and delete entries in database tables. **However, you must remember that Open SQL statements do not check authorization or the consistency of data in the database. The following statements are purely technical means of programming database updates. They are to be used with care, and, outside the SAP transaction concept, only to be used in exceptional cases.**

Note1: Always, All the Database Changing (INSERT, UPDATE, MODIFY,DELETE) Operations for Single Record is through WORK AREA and for Multiple Records is through INTERNAL TABLES.

Note 2 : In All the Database Changing Operations ,

The Structure Of work area <WA> and <ITAB> must be according to the structure of the database table.

INSERT - SYNTAX

Single

INSERT <DBT> FROM <WA>.

Multiple

INSERT <DBT> FROM TABLE <ITAB>
ACCEPTING DUPLICATE KEYS.

Note: ACCEPTING DUPLICATE KEYS Doesn't mean that it Accepts and Inserts Duplicate Records. Instead it Ignore the records which already exists in Database and Inserts the rest of the Records.

Note : Whenever you want to insert more than one line into a database table, it is more efficient to work with an internal table than to insert the lines one by one.

REPORT ZDEMO_OPEN_SQL_INSERT .

```
* PROGRAM      :      ZDEMO_OPEN_SQL_INSERT
* AUTHOR       :      GANAPATI .ADIMULAM
* START DATE   :      27/01/2008
* PURPOSE      :      IS TO WORK WITH ALL OPEN SQL OPERATIONS
* COPIED FROM  :      NA
```

```
* MODIFICATION LOG:
* CHANGE REQUEST :  C11EMAX4756
```

```
*-----
*MOD-001      :      DETAILS OF MODIFICATION 001
*MOD-002      :      DETAILS OF MODIFICATION 002
*SUPPLIER     :      EMAX TECHNOLOGIES,AMEERPET
```

```
TYPES BEGIN OF TY_T001.
INCLUDE STRUCTURE T001.
TYPES END OF TY_T001.
```

```
DATA : IT_T001 TYPE STANDARD TABLE OF TY_T001,
      WA_T001 TYPE TY_T001.
```

```
*FILL 1ST RECORD
WA_T001-BUKRS = 'E001'.
WA_T001-BUTXT = 'eMAX TECHNOLOGIES'.
```

```
APPEND WA_T001 TO IT_T001.
CLEAR WA_T001.
```

```
*FILL 2ND RECORD
WA_T001-BUKRS = 'E002'.
WA_T001-BUTXT = 'CLARION PARK TECHNOLOGIES'.
```

```
APPEND WA_T001 TO IT_T001.
```


CLEAR WA_T001.

*INSERT MULTIPLE RECORDS FROM ITAB
INSERT T001 FROM TABLE IT_T001.

IF SY-SUBRC = 0.

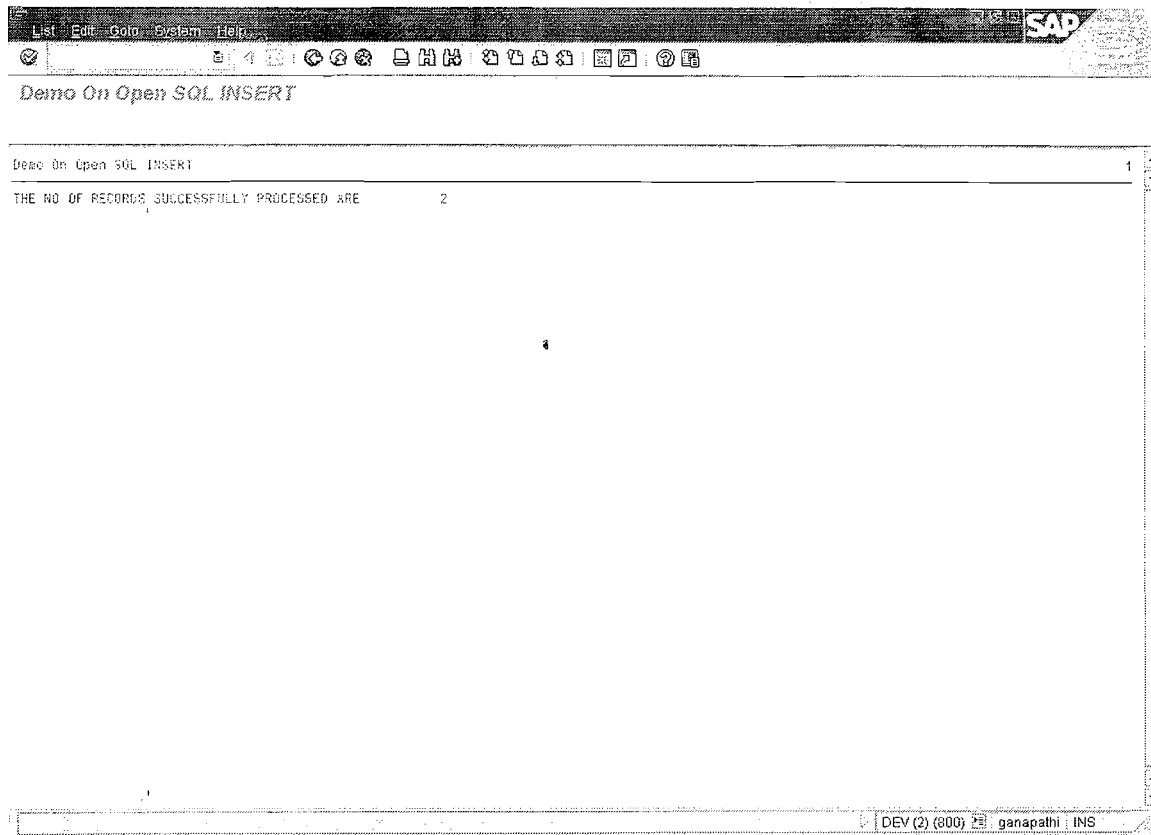
WRITE : 'THE NO OF RECORDS SUCCESSFULLY PROCESSED ARE',
SY-DBCNT.

ELSE.

WRITE : / 'NOT ALL THE RECORDS ARE SUCCESSFULLY PROCESSED'.
ENDIF.

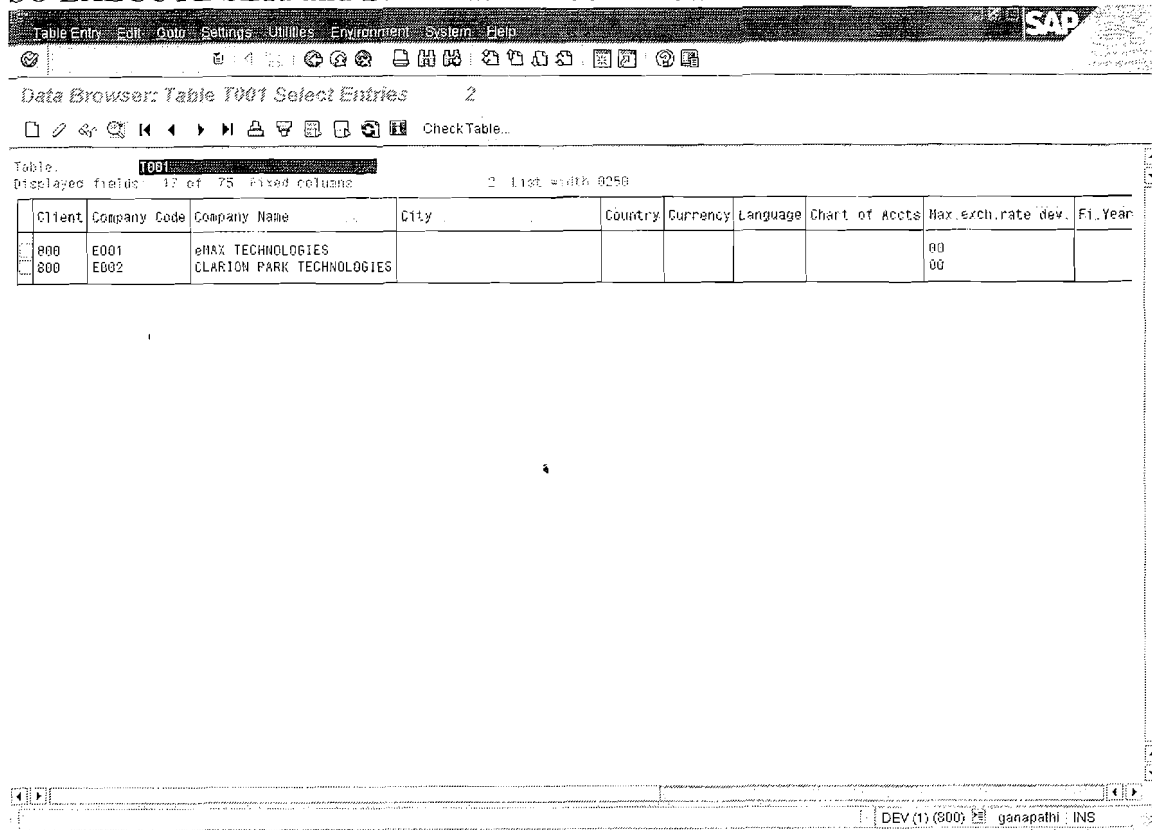
OUTPUT :

EXECUTE THE PROGRAM



NOTE : To Verify the Successful Execution (Insertion Of Records) , Check the Same in Database.

SO EXECUTE SE11 and DISPLAY the Contents From Table T001.



Changing Lines:

UPDATE - Syntax



Column Update

UPDATE <DBT> SET
 <Field1> = <Value1>
 <Field2> = <Value2>...
 WHERE <Condition>.

Row Update

UPDATE <DBT> FROM <WA>. (Single Rec)
 UPDATE <DBT> FROM TABLE <ITAB>.
 (Mult.Rec)

If at least one line is changed, the system sets SY-SUBRC to 0, otherwise to 4.
 SY-DBCNT contains the number of lines changed.

NOTE: Updates (Overwrites) if record exists else Ignores.

Sample Program for Column Update:

REPORT ZDEMO_OPEN_SQL_UPDATE.

```
*****  
* PROGRAM      :      ZDEMO_OPEN_SQL_UPDATE  
* AUTHOR       :      GANAPATI .ADIMULAM  
* START DATE   :      27/01/2008  
* PURPOSE      :      IS TO WORK WITH ALL OPEN SQL OPERATIONS  
* COPIED FROM  :      NA  
*****
```

```
* MODIFICATION LOG:  
* CHANGE REQUEST : C11EMAX4756  
*-----
```

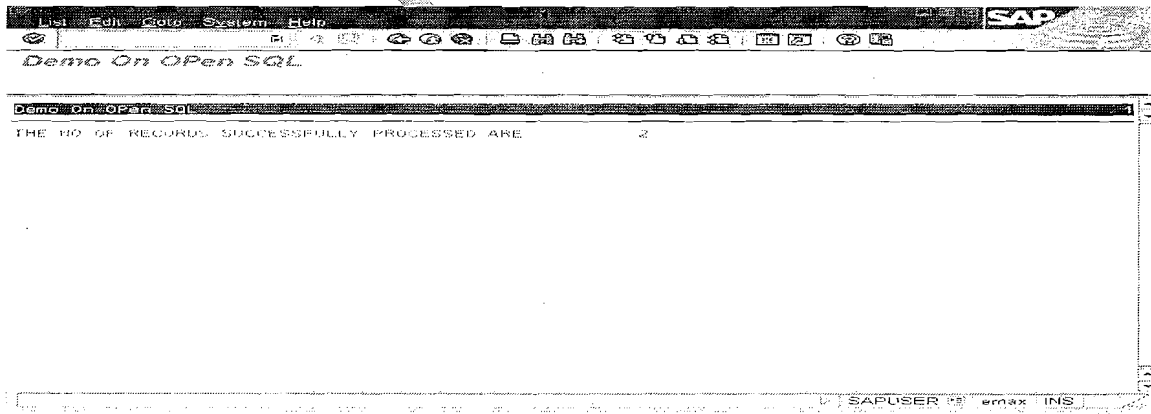
```
*MOD-001      :      DETAILS OF MODIFICATION 001  
*MOD-002      :      DETAILS OF MODIFICATION 002  
*SUPPLIER     :      EMAX TECHNOLOGIES,AMEERPET  
*****
```

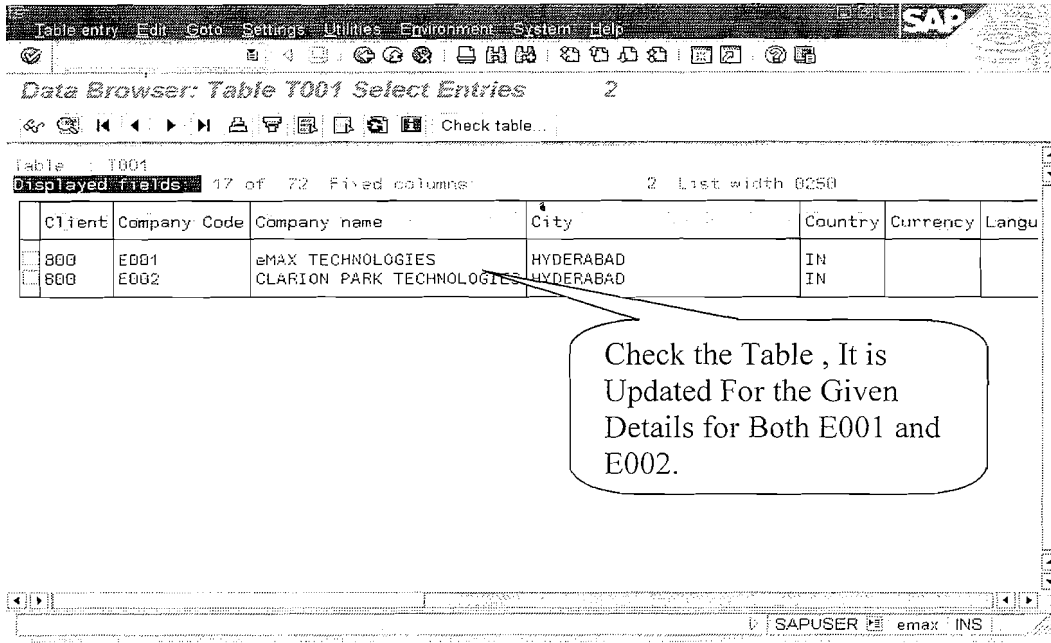
```
*UPDATE COLUMNS ORT01,ORT02.LAND1 FOR THE COMAPNY CODES 'E001',  
'E002'  
UPDATE T001 SET ORT01 = 'HYDERABAD'  
                LAND1 = 'IN'  
                WHERE BUKRS IN ('E001', 'E002').
```

```
WRITE : 'THE NO OF RECORDS SUCCESSFULLY PROCESSED ARE',SY-  
DBCNT.
```

Note : This Program Updates the City and Country Key for the Company Codes 'E001' and 'E002' as we didn't provide the Same at the time of Insertion.

OUTPUT:





Check the Table , It is Updated For the Given Details for Both E001 and E002.

MODIFY (Inserting or Changing) Lines:

MODIFY – Syntax

Acts As

INSERT When Record Doesn't Exist

UPDATE When Record Exist.

Syntax For Single Record : MODIFY <DBT> FROM <WA>.

For Multiple Recs : MODIFY <DBT> FROM TABLE <Itab>.

EXAMPLE :

REPORT ZDEMO_OPEN_SQL_MODIFY .

```

*****
** PROGRAM      : ZDEMO_OPEN_SQL_MODIFY
* AUTHOR       : GANAPATI .ADIMULAM
* START DATE   : 30/06/2006
* PURPOSE      : IS TO WORK WITH MODIFY
* COPIED FROM  : NA
*****
* MODIFICATION LOG:
* CHANGE REQUEST : C11EMAX4756
*
*-----
*MOD-001      : DETAILS OF MODIFICATION 001
*MOD-002      : DETAILS OF MODIFICATION 002
*SUPPLIER     : EMAX TECHNOLOGIES,AMEERPET
    
```

TYPES BEGIN OF TY_T001.
INCLUDE STRUCTURE T001.
TYPES END OF TY_T001.

DATA : IT_T001 TYPE STANDARD TABLE OF TY_T001,
WA_T001 TYPE TY_T001.

WA_T001-BUKRS = 'E003'.
WA_T001-BUTXT = 'AG TECHNOLOGIES'.
WA_T001-ORT01 = 'MUMABI'.

MODIFY T001 FROM WA_T001.

WRITE : 'THE NO OF RECORDS SUCCESSFULLY PROCESSED ARE',SY-
DBCNT.

CLEAR WA_T001.

WA_T001-BUKRS = 'E004'.
WA_T001-BUTXT = 'ASAP TECHNOLOGIES'.
WA_T001-ORT01 = 'CHENNAI'.

APPEND WA_T001 TO IT_T001.
CLEAR WA_T001.

WA_T001-BUKRS = 'E005'.
WA_T001-BUTXT = 'IFLEX TECHNOLOGIES'.
WA_T001-ORT01 = 'HYDERABAD'.

APPEND WA_T001 TO IT_T001.
CLEAR WA_T001.

WA_T001-BUKRS = 'E002'.
WA_T001-BUTXT = 'CLARION PARK TECHNOLOGIES'.
WA_T001-ORT01 = 'BANGLORE'.
WA_T001-WAERS = 'INR'.

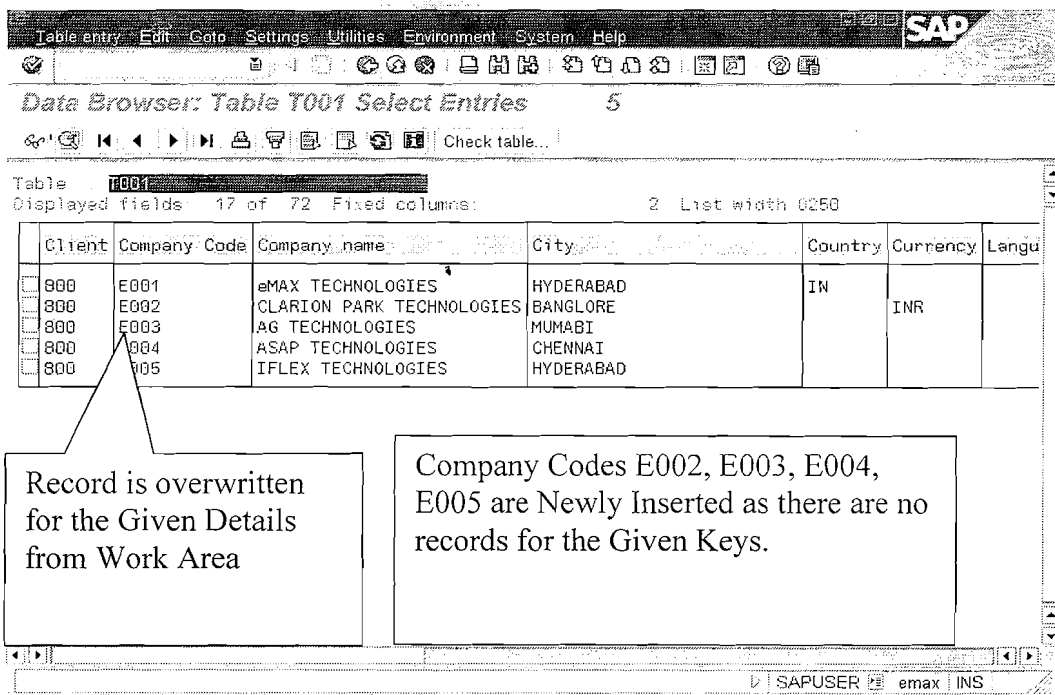
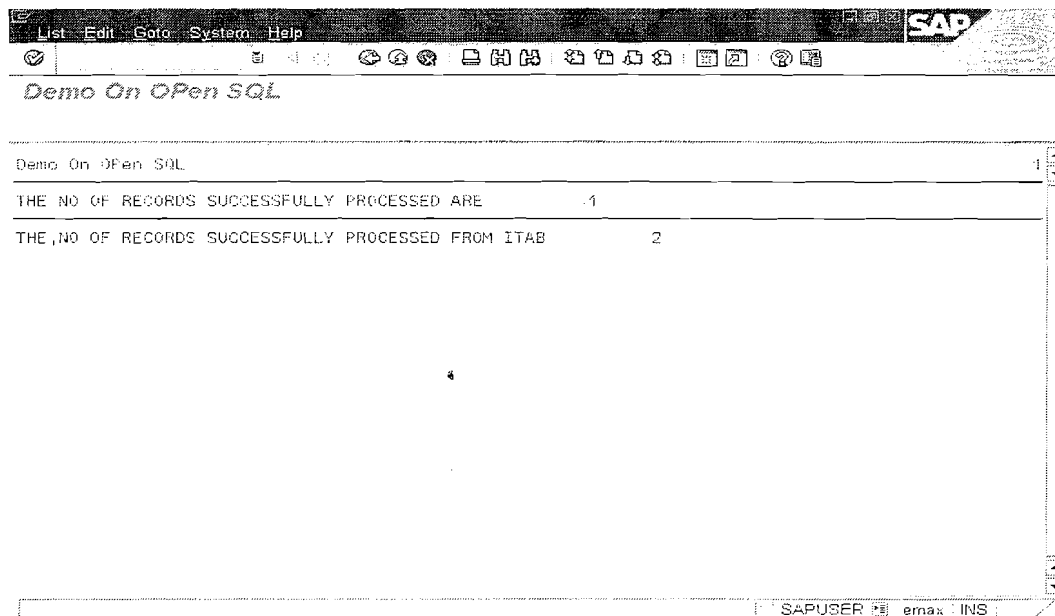
APPEND WA_T001 TO IT_T001.

*MODIFY MULTIPLE RECORDS FROM ITAB.
MODIFY T001 FROM TABLE IT_T001.

ULINE.
WRITE : 'THE NO OF RECORDS SUCCESSFULLY PROCESSED FROM
ITAB',

SY-DBCNT.

OUTPUT : EXECUTE THE PROGRAM



Deleting Lines :

DELETE <DBT> FROM <WA>. "Single Rec
DELETE <DBT> FROM TABLE <Itab>. "Multiple Rec

DELETE FROM <DBT> WHERE <Condition>.

Note : Deletion Depends On the WHERE Condition.

Note : Deletes If Exists Else Ignores.

```
REPORT ZDEMO_OPEN_SQL_DELETE .
*****
* PROGRAM      :      ZDEMO_OPEN_SQL_DELETE
* AUTHOR       :      GANAPATI .ADIMULAM
* START DATE   :      27/01/2008
* PURPOSE      :      IS TO WORK WITH ALL OPEN SQL
OPERATIONS
* COPIED FROM  :      NA
*****
* MODIFICATION LOG:
* CHANGE REQUEST :  C11EMAX4756
*-----
*MOD-001      :      DETAILS OF MODIFICATION 001
*MOD-002      :      DETAILS OF MODIFICATION 002
*SUPPLIER     :      EMAX TECHNOLOGIES,AMEERPET
*****
TYPES BEGIN OF TY_T001.
INCLUDE STRUCTURE T001.
TYPES END OF TY_T001.

DATA :IT_T001 TYPE STANDARD TABLE OF TY_T001,
      WA_T001 TYPE TY_T001.

WA_T001-BUKRS = 'E004'.
WA_T001-BUTXT = 'ASAP TECHNOLOGIES'.
WA_T001-ORT01 = 'CHENNAI'.

APPEND WA_T001 TO IT_T001.
CLEAR WA_T001.

WA_T001-BUKRS = 'E005'.
WA_T001-BUTXT = 'IFLEX TECHNOLOGIES'.
WA_T001-ORT01 = 'HYDERABAD'.

APPEND WA_T001 TO IT_T001.
```

*DELETE MULTIPLE RECORDS FOR THE ENTRIES IN ITAB.
DELETE T001 FROM TABLE IT_T001.

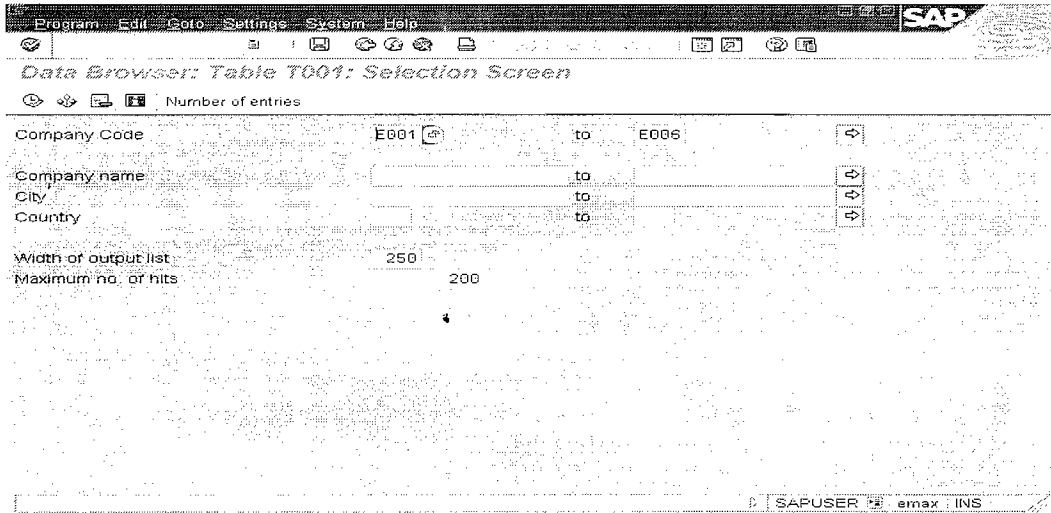
ULINE.

WRITE : 'THE NO OF RECORDS SUCCESSFULLY DELETED',SY-DBCNT.

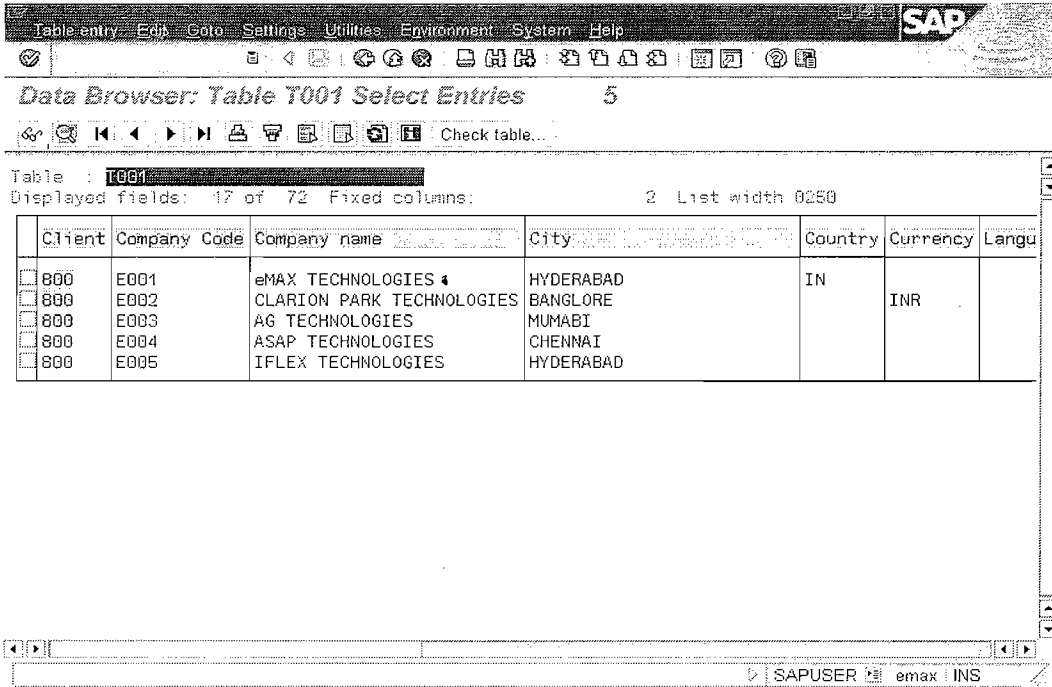
NOTE : EXECUTE SE11 and Check For the Contents Before AND After

Executing the Program.

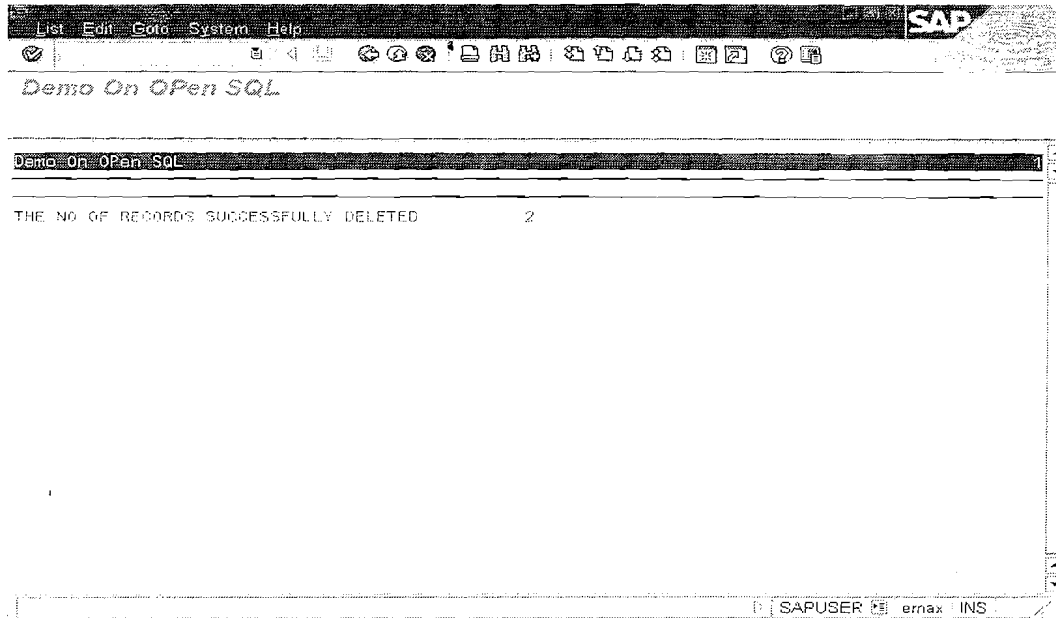
BEFORE :



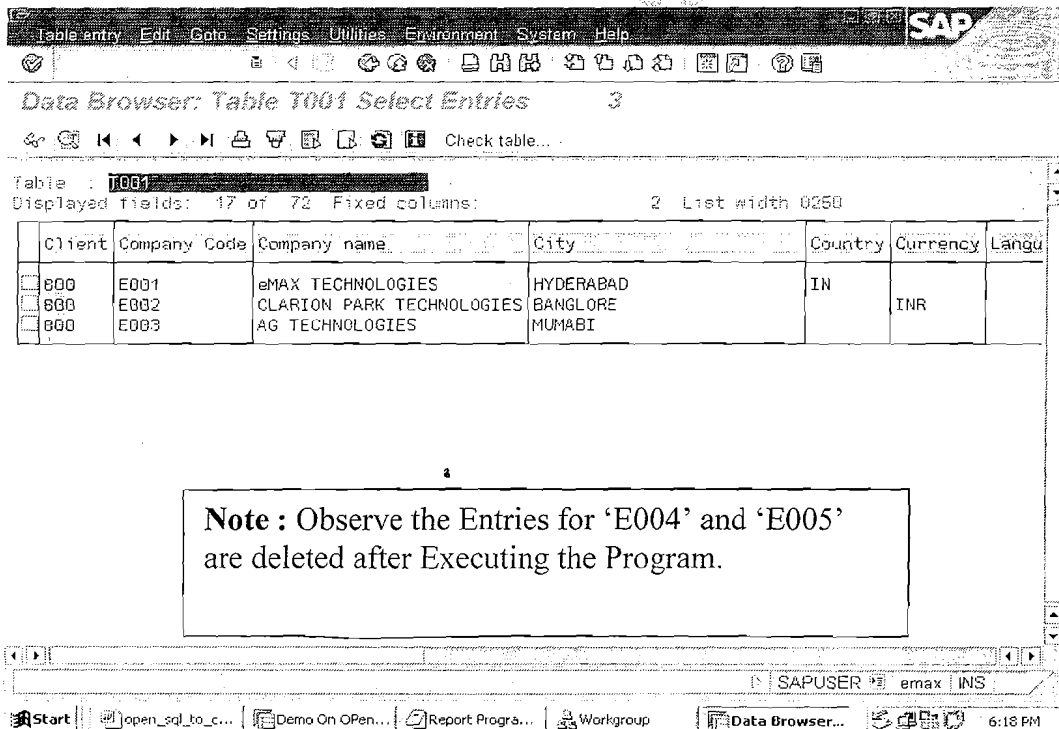
Entries in T001 Before Executing(Deleting the Entries E004,E005) the Program.



OUTPUT : EXECUTE THE PROGRAM



Entries in Database Table , After Executing the Program :



Result Of Database Changing Operations:

	SY-SUBRC For Single Record	SY-SUBRC For Multiple Records
INSERT	0 - For Success Insertion 4 - For Un Successful	0 - For Successful Insertion of All Records. 4 - When One More Lines Cannot be Inserted Because of the Duplicates.
UPDATE COLUMNS	UPDATE <target> SET <set₁> <set₂> ... [WHERE <cond>]. 0 - If atleast One Line is Changed. 4 - When None of the Records is Changed.	0 - If atleast One Line is Changed. 4 - When None of the Records is Changed.
UPDATE(Overwritin g)	0 - For Success UPDATE	0 - For Successful Updation Of all Records 4 - Otherwise
MODIFY	SY-SUBRC is always set to 0. Because it Overwrites the Record if it finds a Match with Primary Key Otherwise it Inserts a New Record. So it is Always Successful.	
Note : SY-DBCNT Always returns the number of Records Successfully processed.		

Working With Database Reading :**DAY-2**

The Open SQL statement for reading data from database tables is:

```
SELECT <F1>
      <F2>
      .
      .
      <Fn>
INTO  TABLE <Itab>
FROM   <DBT>
WHERE  <Condition>.
```

NOTE : We Can also Use **GROUP BY, HAVING, ORDER BY** . But it is not recommended in Real time Because we have lots of records , so it is not better to do it at Database Level.

Note : Make Sure that the Structure Of the Internal Table and the Order of Fields in the **SELECTs** Should be always Same Because the data is transferred according i.e 1st Field Content Of **SELECT** Into 1st Field Of **<ITAB>** and Similarly all Other Fields.

Note : When we need to read 80% of the Fields From a table then it is better to declare the Target Area with all the Fields and Query for all the Columns (**SELECT ***).

Reading Aggregate Data for Columns

- **MAX:** returns the maximum value of the column **<C_i>**
- **MIN:** returns the minimum value of the column **<C_i>**
- **AVG:** returns the average value of the column **<C_i>**
- **SUM:** returns the sum value of the column **<C_i>**
- **COUNT:** counts values or lines as follows:
 - **COUNT (DISTINCT <C_i>)** returns the number of different values in the column **<C_i>**.
 - **COUNT(*)** returns the total number of lines in the selection.

You can exclude duplicate values from the calculation using the **DISTINCT** option. The spaces between the parentheses and the arguments of the aggregate expressions must not be left out. **The arithmetic operators AVG and SUM only work with numeric fields.**

EXAMPLE Program :

REPORT ZDEMO_GET_COMPANY_DATA.

```
* PROGRAM      :      ZDEMO_GET_COMPANY_DATA
* AUTHOR       :      GANAPATI .ADIMULAM
* START DATE   :      27/01/2008
* PURPOSE      :      IS TO WORK WITH INTO STATEMENT IN
                    SELECT
* COPIED FROM  :      NA
```

* MODIFICATION LOG:

* CHANGE REQUEST : C11EMAX4756

*-----

```
*MOD-001      :      DETAILS OF MODIFICATION 001
*MOD-002      :      DETAILS OF MODIFICATION 002
*SUPPLIER     :      EMAX TECHNOLOGIES,AMEERPET
```

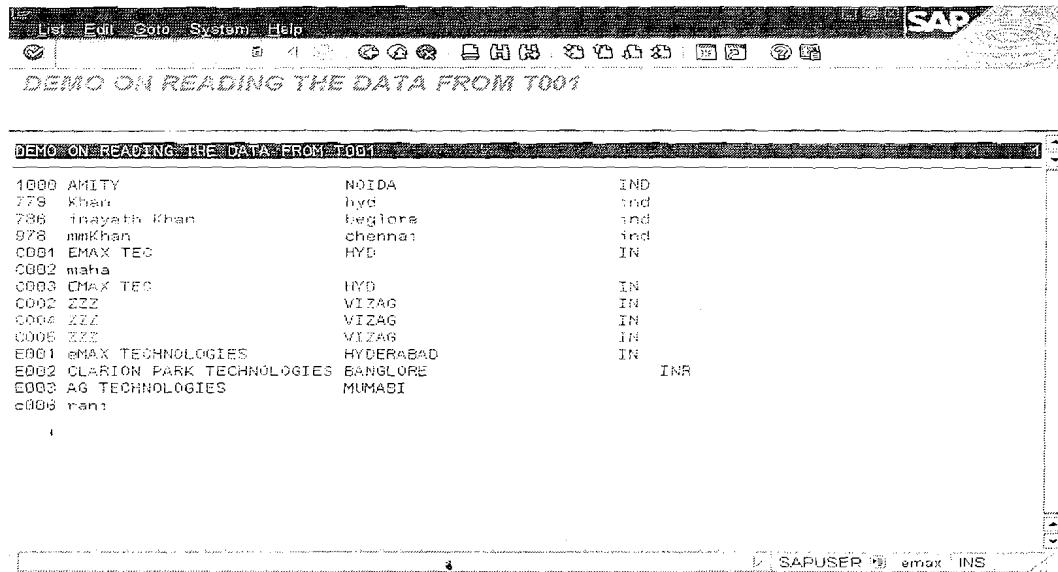
```
TYPES : BEGIN OF TY_T001,
        BUKRS TYPE BUKRS, "COMPANY CODE
        BUTXT TYPE BUTXT, "NAME
        ORT01 TYPE ORT01, "CITY
        LAND1 TYPE LAND1, "COUNTRY KEY
        WAERS TYPE WAERS, "CURRENCY KEY
END OF TY_T001.
```

```
DATA : IT_T001 TYPE STANDARD TABLE OF TY_T001,
        WA_T001 TYPE TY_T001.
```

```
SELECT BUKRS
        BUTXT
        ORT01
        LAND1
        WAERS
INTO TABLE IT_T001
FROM T001 UP TO 20 ROWS.
```

```
*DISPLAY DATA
LOOP AT IT_T001 INTO WA_T001.
  WRITE : /
          WA_T001-BUKRS,
          WA_T001-BUTXT,WA_T001-ORT01,WA_T001-LAND1,
          WA_T001-WAERS.
CLEAR WA_T001.
ENDLOOP.
```

Out Put: EXECUTE THE PROGRAM



Note : This Reads all the 20 rows at a time into the internal table But there is another way where you can read the data in packages of records(i.e <N> no of records every time).

REPORT ZDEMO_GET_COMPANY_DATA .

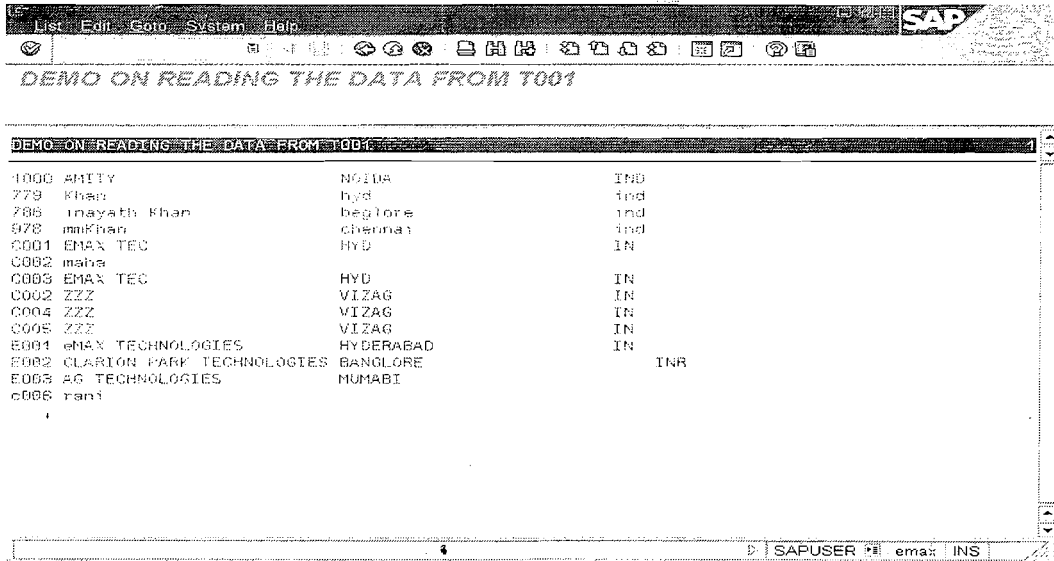
```

*****
* PROGRAM       :      ZDEMO_GET_COMPANY_CODE
* AUTHOR        :      GANAPATI.ADIMULAM
* START DATE   :      31/01/2008
* PURPOSE      :      IS TO WORK WITH APPENDING STATEMENT
                   IN SELECT
* COPIED FROM  :      NA
*****
* MODIFICATION LOG:
* CHANGE REQUEST :  C11EMAX4756
*-----
*MOD-001       :      DETAILS OF MODIFICATION 001
*MOD-002       :      DETAILS OF MODIFICATION 002
*SUPPLIER      :      EMAX TECHNOLOGIES,AMEERPET
*****
TYPES : BEGIN OF TY_T001,
        BUKRS TYPE BUKRS, "COMPANY CODE
        BUTXT TYPE BUTXT, "NAME
        ORT01 TYPE ORT01, "CITY
        LAND1 TYPE LAND1, "COUNTRY KEY
        WAERS TYPE WAERS, "CURRENCY KEY
        . END OF TY_T001.
    
```

```
DATA WA_T001 TYPE TY_T001.
DATA : IT_T001 TYPE STANDARD TABLE OF TY_T001.
```

```
SELECT BUKRS
       BUTXT
       ORT01
       LAND1
       WAERS
APPENDING TABLE IT_T001
FROM T001 UP TO 20 ROWS
PACKAGE SIZE 5.
ENDSELECT.
*DISPLAY DATA
LOOP AT IT_T001 INTO WA_T001.
  WRITE : /
         WA_T001-BUKRS,WA_T001-BUTXT,
         WA_T001-ORT01,WA_T001-LAND1,
         WA_T001-WAERS.
CLEAR WA_T001.
ENDLOOP.
```

OutPut :



Note : It Reads the 20 rows from T001 But Every time it reads and appends Only 5 records.

Working With JOINS:

In a relational database, you normally need to read data **simultaneously from more than one database table into an application program**. You can read from more than one table in a single SELECT statement via **JOINS**.

Note : There are TWO types Of JOINS.

- A) INNER JOIN
- B) LEFT OUTER JOIN

Syntax : SELECT <DBT1>~<F1>
 <DBT1>~<F2>
 <DBT1>~<F3>
 <DBT1>~<F4>
 <DBT2>~<F5>
 <DBT2>~<F6>
 <DBT2>~<F7>

INTO TABLE <Itab>

FROM <DBT1> INNER JOIN / LEFT OUTER JOIN

ON <DBT1>~<F1> = <DBT2>~<F2> "JOIN CONDITION

WHERE (Conditions On Fields From DBT1 and DBT2 If Required).

Note : When Working With JOINS Make Sure that <DBT1> (Left Hand Side Table) is Always One Entry Table and <DBT2> is Always Many Entries Table(Right Hand Side Table).

INNER JOIN: The Join Expression links each line of <DBT1> with the lines in <DBT2> that meet the Join condition <cond>. This means that there is always one or more lines from the right-hand table that is linked to each line from the left-hand table by the join. **If <DBT2> does not contain any lines that meet the JOIN condition <cond>, the line from <DBT1> is not included in the selection.**

Left Outer Join :

In an inner join, a line from the left-hand database table or join is only included in the selection if there is one or more lines in the right-hand database table that meet the ON condition <cond>. **The left outer join, on the other hand, reads lines from the left-hand database table or join even if there is no corresponding line in the right-hand table.**

Note : The Same Syntax rules apply as in an inner join. The tables are linked in the same way as the inner join with the one exception that **all lines** selected from

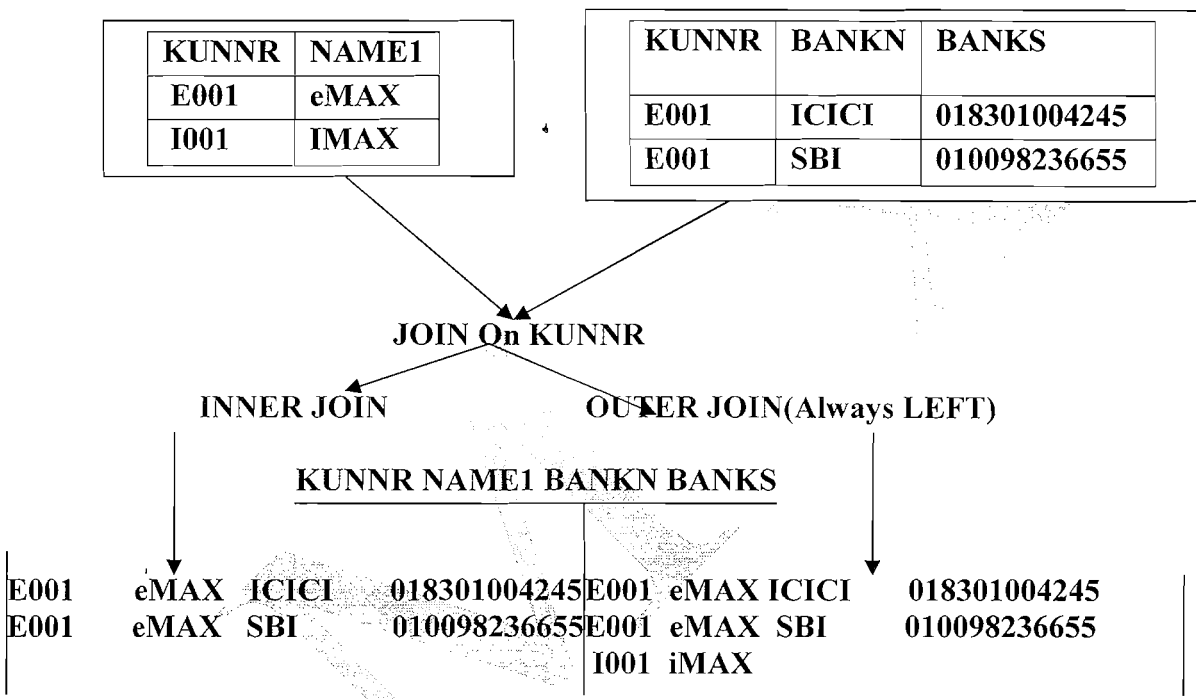
<DBT1> are included in the final selection. Even <DBT2> does not contain any lines that meet the condition <Condition>, i.e. **the system includes a single line From <DBT1> in the selection whose columns from <DBT2> are filled with null values.**

Example :

Example for Both Inner and Outer Join :

Entries from KNA1

Entries From KNBK(Bank Details).



EXAMPLE PROGRAM ON JOINS :

```

*****
* PROGRAM       :      ZDEMO_JOINS
* AUTHOR        :      GANAPATI . ADIMIULAM
* START DATE    :      27/01/2008
* PURPOSE       :      IS TO WORK WITH JOINS FROM TWO TABLE
* COPIED FROM   :      NA
*****
* MODIFICATION LOG:
* CHANGE REQUEST :      C11EMAX4756
* -----
* MOD-001       :      DETAILS OF MODIFICATION 001
* MOD-002       :      DETAILS OF MODIFICATION 002
* SUPPLIER      :      EMAX TECHNOLOGIES,AMEERPET
    
```

REPORT ZGDEMO_JOINS NO STANDARD PAGE HEADING LINE-SIZE 150
LINE-COUNT 25.

DATA : V_KUNNR TYPE KUNNR.

TYPES : BEGIN OF TY_SALES,

 KUNNR TYPE KUNNR, "CUSTOMER NOI

 VBELN TYPE VBELN_VA, "SALES DOCUMENT

END OF TY_SALES.

DATA WA_SALES TYPE TY_SALES.

DATA : IT_SALES TYPE TABLE OF TY_SALES.

SELECTION-SCREEN BEGIN OF BLOCK B1 WITH FRAME TITLE TEXT-000.

 PARAMETER : RB_IJOIN RADIOBUTTON GROUP G1,

 RB_OJOIN RADIOBUTTON GROUP G1.

SELECTION-SCREEN END OF BLOCK B1.

SELECTION-SCREEN BEGIN OF BLOCK B2 WITH FRAME TITLE TEXT-001.

SELECT-OPTIONS S_KUNNR FOR V_KUNNR.

SELECTION-SCREEN END OF BLOCK B2.

* START-OF-SELECTION. *

START-OF-SELECTION.

IF RB_IJOIN = 'X'.

WRITE : / 'THE RESULT OF INNER JOIN IS'.

SELECT

 VBAK~KUNNR

 VBAK~VBELN

INTO TABLE IT_SALES

FROM KNA1 INNER JOIN VBAK

ON KNA1~KUNNR = VBAK~KUNNR

WHERE KNA1~KUNNR IN S_KUNNR.

ELSEIF RB_OJOIN = 'X'.

SELECT

 VBAK~KUNNR

 VBAK~VBELN

INTO TABLE IT_SALES

FROM KNA1 LEFT OUTER JOIN VBAK

ON KNA1~KUNNR = VBAK~KUNNR

WHERE KNA1~KUNNR IN S_KUNNR.

```
WRITE : / 'THE RESULT OF LEFT OUTER JOIN IS'.  
ENDIF.
```

```
IF SY-SUBRC NE 0.
```

```
  WRITE : / 'NO RECORDS FOUND FOR THE GIVEN SELECTION CRITERIA'.
```

```
  EXIT.
```

```
ELSE.
```

```
DATA V_LINES TYPE I.
```

```
SORT IT_SALES BY KUNNR VBELN.
```

```
DESCRIBE TABLE IT_SALES LINES V_LINES.
```

```
WRITE : / 'THE NO OF RECORDS FROM IT_SALES', V_LINES.
```

```
ULINE.
```

```
WRITE: / 'CUSTOMER NO', 'SALES ORDER'.
```

```
ULINE.
```

```
  LOOP AT IT_SALES INTO WA_SALES.
```

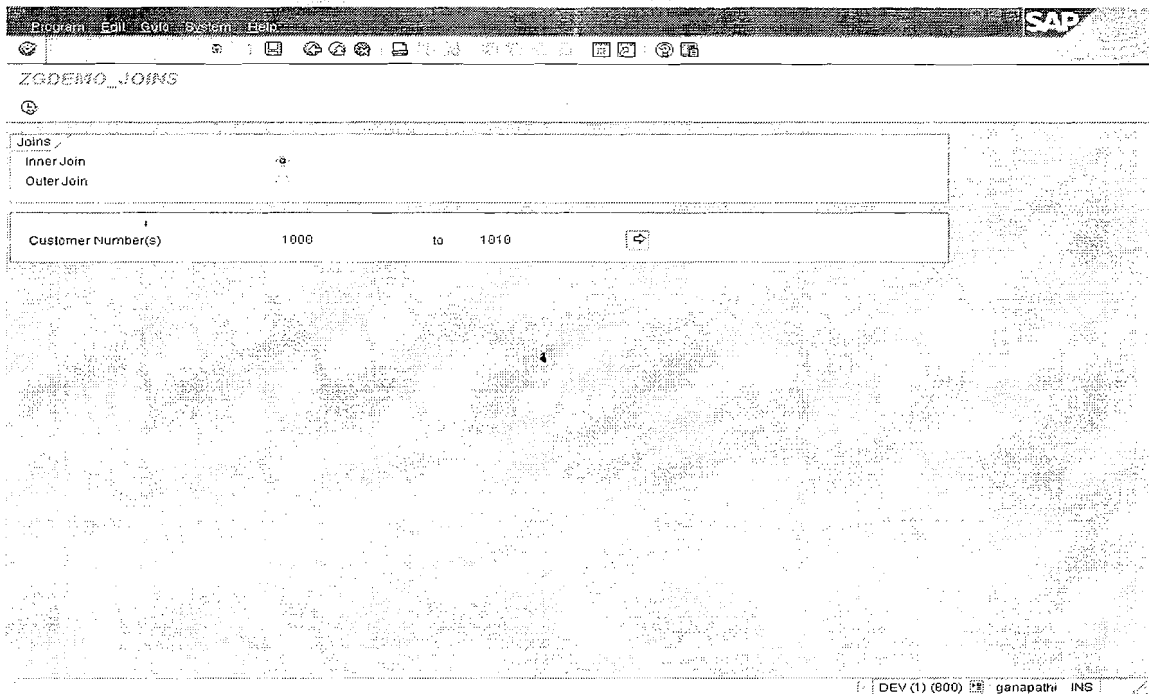
```
    WRITE : / WA_SALES-KUNNR,
```

```
            WA_SALES-VBELN.
```

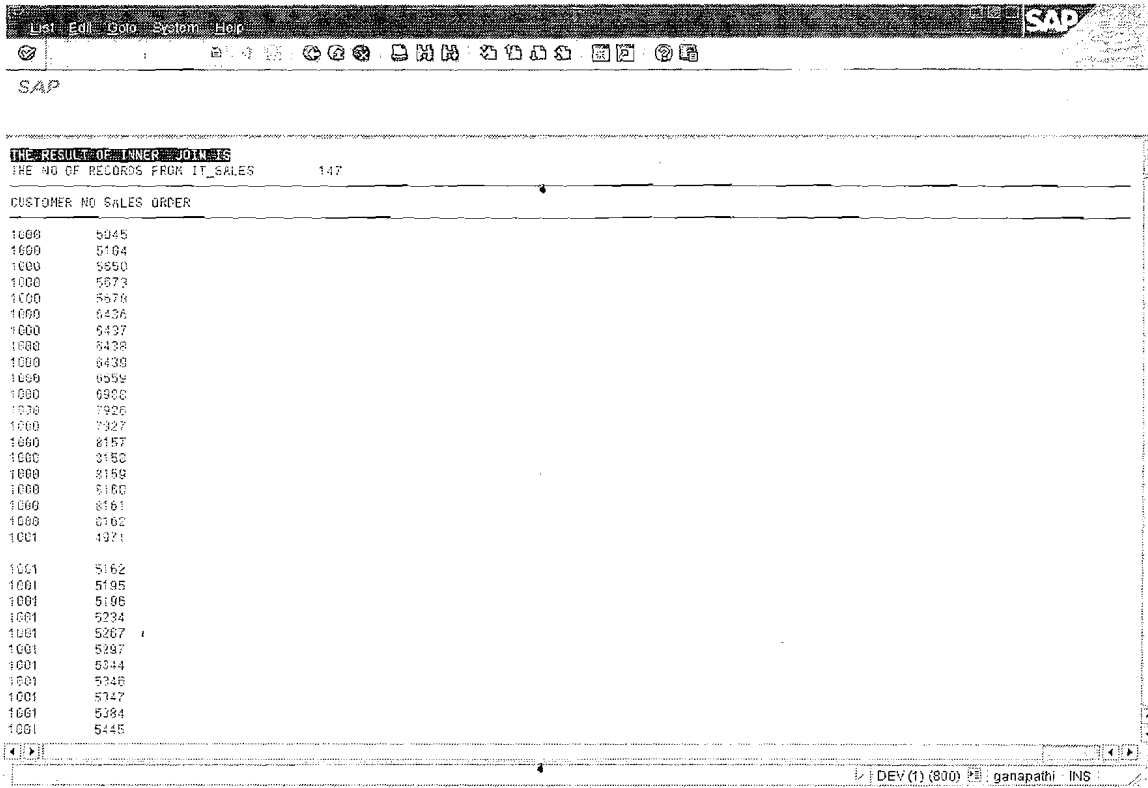
```
  ENDLOOP.
```

```
ENDIF.
```

OUTPUT:- EXECUTE THE PROGRAM FOR INNER JOIN



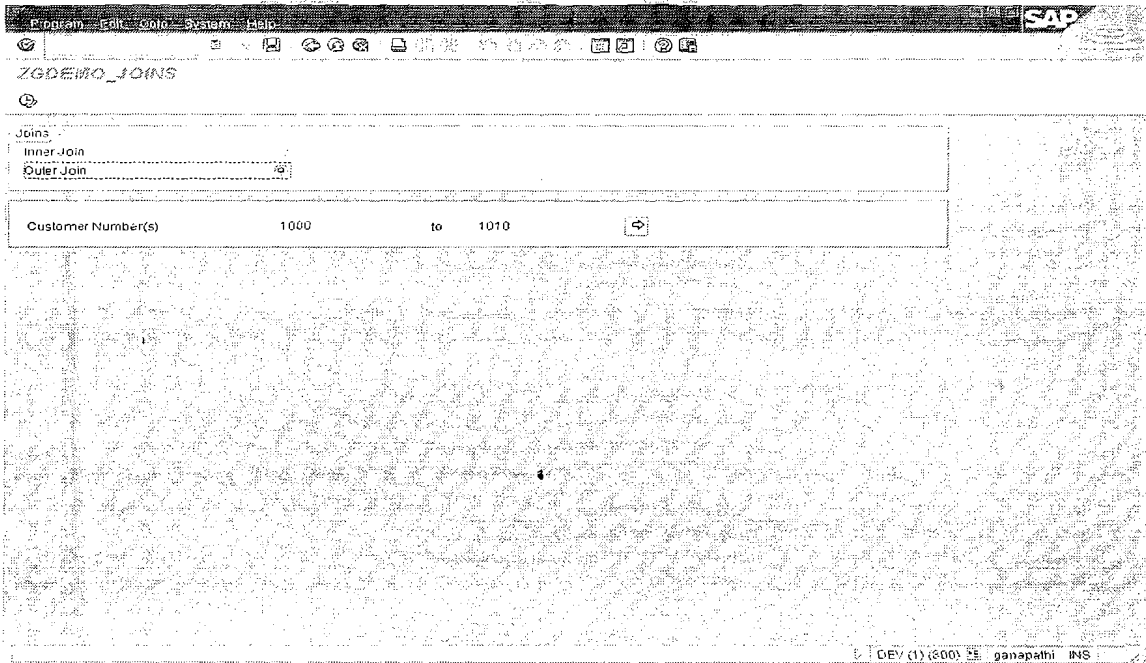
Press F8 or click on Execute button. This is the final output screen.



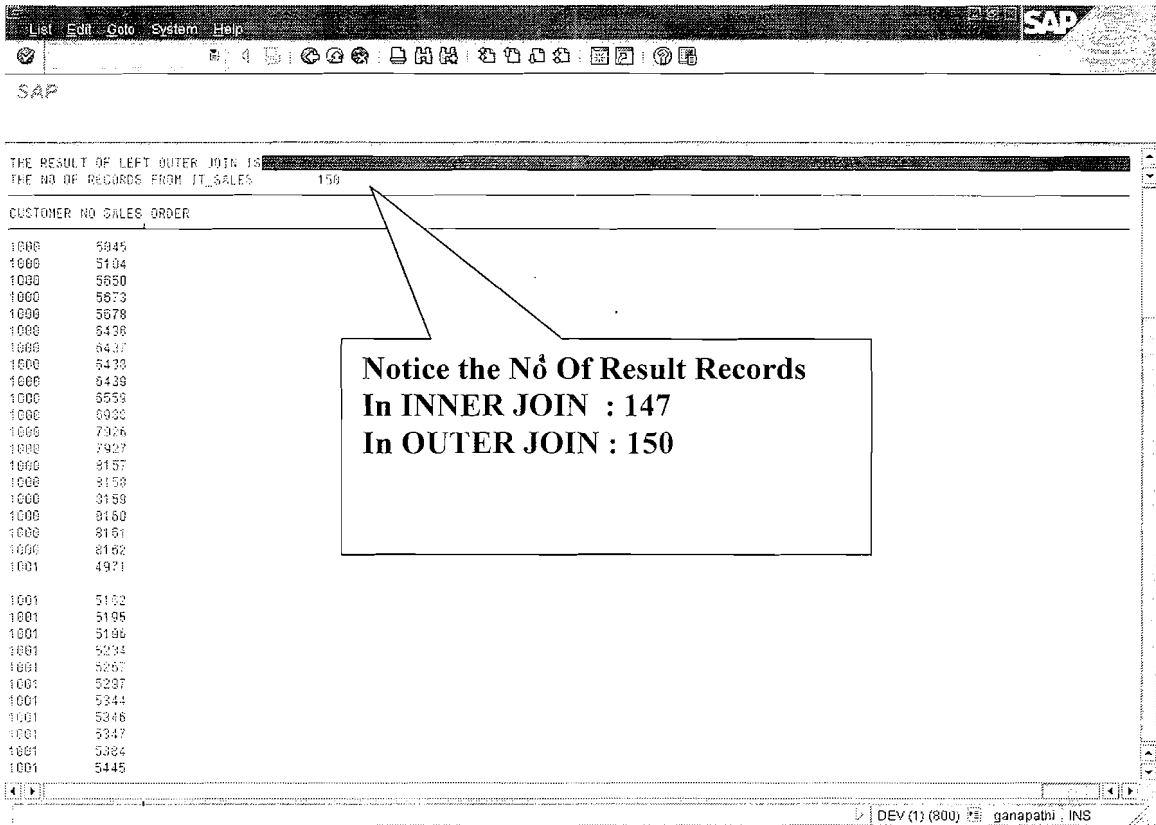
The screenshot shows the SAP Open SQL interface. The title bar reads "THE RESULT OF INNER JOINS". Below the title, it says "THE NO OF RECORDS FROM IT_SALES" followed by the number "147". The main area contains a table with the following columns: "CUSTOMER NO", "SALES ORDER", and "ORDER". The table lists 147 records, each with a customer number and a sales order number. The status bar at the bottom indicates "DEV (1) (800) ganapathi INS".

CUSTOMER NO	SALES ORDER
1000	5045
1000	5104
1000	5050
1000	5073
1000	5076
1000	5436
1000	5437
1000	5438
1000	5438
1000	5059
1000	5900
1000	7926
1000	7927
1000	8157
1000	8150
1000	8158
1000	8160
1000	8161
1000	8162
1001	4371
1001	5162
1001	5195
1001	5106
1001	5234
1001	5267
1001	5287
1001	5344
1001	5346
1001	5347
1001	5384
1001	5445

EXECUTE THE PROGRAM FOR OUTER JOIN :



The screenshot shows the SAP Open SQL program configuration screen. The title bar reads "ZGDENRQ_JOINS". The "Joins" section is expanded, showing "Outer Join" selected. Below this, there is a field for "Customer Number(s)" with the value "1000" and a range indicator "to 1010". The status bar at the bottom indicates "DEV (1) (800) ganapathi INS".



THE RESULT OF LEFT OUTER JOIN IS
THE NO OF RECORDS FROM IT_SALES 150

CUSTOMER NO SALES ORDER

1000	5945
1000	5104
1000	5650
1000	5673
1000	5678
1000	6436
1000	6437
1000	6438
1000	6439
1000	6559
1000	6900
1000	7306
1000	7927
1000	8157
1000	8158
1000	8159
1000	8160
1000	8161
1000	8162
1001	4971
1001	5102
1001	5195
1001	5196
1001	5234
1001	5257
1001	5297
1001	5344
1001	5346
1001	5347
1001	5384
1001	5445

**Notice the No Of Result Records
In INNER JOIN : 147
In OUTER JOIN : 150**

DEV (1) (800) ganapati INS

Working With FOR ALL ENTRIES:

NOTE : FOR ALL ENTRIES CAN REPLACE INNER JOINS.

IS MANDATORY TO READ THE DATA FROM DATABASE AND
INTERNAL TABLE COMBINATION.

The WHERE clause of the SELECT statement has a special variant that allows you to derive conditions from the lines and columns of an internal table:

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

<cond> may be formulated as described above. If you specify a field of the internal table <itab> as an operand in a condition, you address all lines of the internal table. The comparison is then performed for each line of the internal table. For each line, the system selects the lines from the database table that satisfy the condition. **The result set of the SELECT statement is the union of the individual selections for each line of the internal table.**

You can use the option FOR ALL ENTRIES to replace **nested select loops** by operations on internal tables. **This can significantly improve the performance for large sets of selected data.**

REPORT ZGDemo_FOR_ALL_ENTRIES line-size 300.

```

*****
* PROGRAM      :      ZGDemo_FOR_ALL_ENTRIES
* AUTHOR       :      GANAPATI . ADIMULAM
* START DATE   :      28/01/2008
* PURPOSE      :      IS TO WORK WITH FOR ALL ENTRIES
* COPIED FROM  :      NA
*****
* MODIFICATION LOG:
* CHANGE REQUEST :  C11EMAX4756
*-----
*MOD-001      :      DETAILS OF MODIFICATION 001
*MOD-002      :      DETAILS OF MODIFICATION 002
*SUPPLIER     :      EMAX TECHNOLOGIES,AMEERPET
*****

```

REPORT ZGDEMO_FOR_ALL_ENTRIES

DATA : V_BELNR TYPE BELNR_D.
 SELECT-OPTIONS : S_BELNR FOR V_BELNR.

TYPES : BEGIN OF TY_BKPF,
 BUKRS TYPE BUKRS, " COMPANY CODE
 BELNR TYPE BELNR_D, " ACCOUNTING DOCUMENT NUMBER
 XBLNR TYPE XBLNR, " REFERENCE DOCUMENT NUMBER
 BUDAT TYPE BUDAT, " POSTING DATE
 END OF TY_BKPF.

TYPES : BEGIN OF TY_BSEG,
 BUKRS TYPE BUKRS, " COMPANY CODE
 BELNR TYPE BELNR_D, " ACCOUNTING DOCUMENT NUMBER
 BUZEI TYPE BUZEI, " LINE ITEM NUMBER
 GSBER TYPE GSBER, " BUSINESS AREA
 ZUONR TYPE DZUONR, " ASSIGNMENT NUMBER
 WRBTR TYPE WRBTR, " AMOUNT IN CURRENCY
 KUNNR TYPE KUNNR, " CUSTOMER NUMBER
 END OF TY_BSEG.

*DECLARING A FINAL STRUCTURE FOR BSEG AND BKPF.

TYPES : BEGIN OF TY_FINAL,
 BUKRS TYPE BUKRS, " COMPANY CODE
 BELNR TYPE BELNR_D, " ACCOUNTING DOCUMENT NUMBER
 XBLNR TYPE XBLNR, " REFERENCE DOCUMENT NUMBER
 BUDAT TYPE BUDAT, " POSTING DATE
 BUZEI TYPE BUZEI, " LINE ITEM NUMBER
 GSBER TYPE GSBER, " BUSINESS AREA
 ZUONR TYPE DZUONR, " ASSIGNMENT NUMBER
 WRBTR TYPE WRBTR, " AMOUNT IN CURRENCY
 KUNNR TYPE KUNNR, " CUSTOMER NUMBER
 END OF TY_FINAL.

 * INTERNAL TABLES & WORK AREAs *

DATA : IT_BSEG TYPE STANDARD TABLE OF TY_BSEG,
 IT_BKPF TYPE STANDARD TABLE OF TY_BKPF,
 IT_FINAL TYPE STANDARD TABLE OF TY_FINAL.

DATA : WA_BSEG TYPE TY_BSEG,
 WA_BKPF TYPE TY_BKPF,
 WA_FINAL TYPE TY_FINAL.

```
SELECT BUKRS
  BELNR
  XBLNR
  BUDAT FROM BKPF
 INTO TABLE IT_BKPF
 WHERE BELNR IN S_BELNR.
```

```
IF NOT IT_BKPF IS INITIAL.
SELECT BUKRS
  BELNR
  BUZEI
  GSBER
  ZUONR
  WRBTR
  KUNNR FROM BSEG
 INTO TABLE IT_BSEG
 FOR ALL ENTRIES IN IT_BKPF
 WHERE BELNR = IT_BKPF-BELNR.
ENDIF.
```

```
LOOP AT IT_BSEG INTO WA_BSEG.
MOVE-CORRESPONDING : WA_BSEG TO WA_FINAL.
READ TABLE IT_BKPF INTO WA_BKPF WITH KEY BUKRS = WA_FINAL-
BUKRS
      BELNR = WA_FINAL-BELNR.
```

```
IF SY-SUBRC = 0.
MOVE : WA_BKPF-XBLNR TO WA_FINAL-XBLNR,
      WA_BKPF-BUDAT TO WA_FINAL-BUDAT.
ENDIF.
APPEND WA_FINAL TO IT_FINAL.
CLEAR WA_FINAL.
ENDLOOP.
```

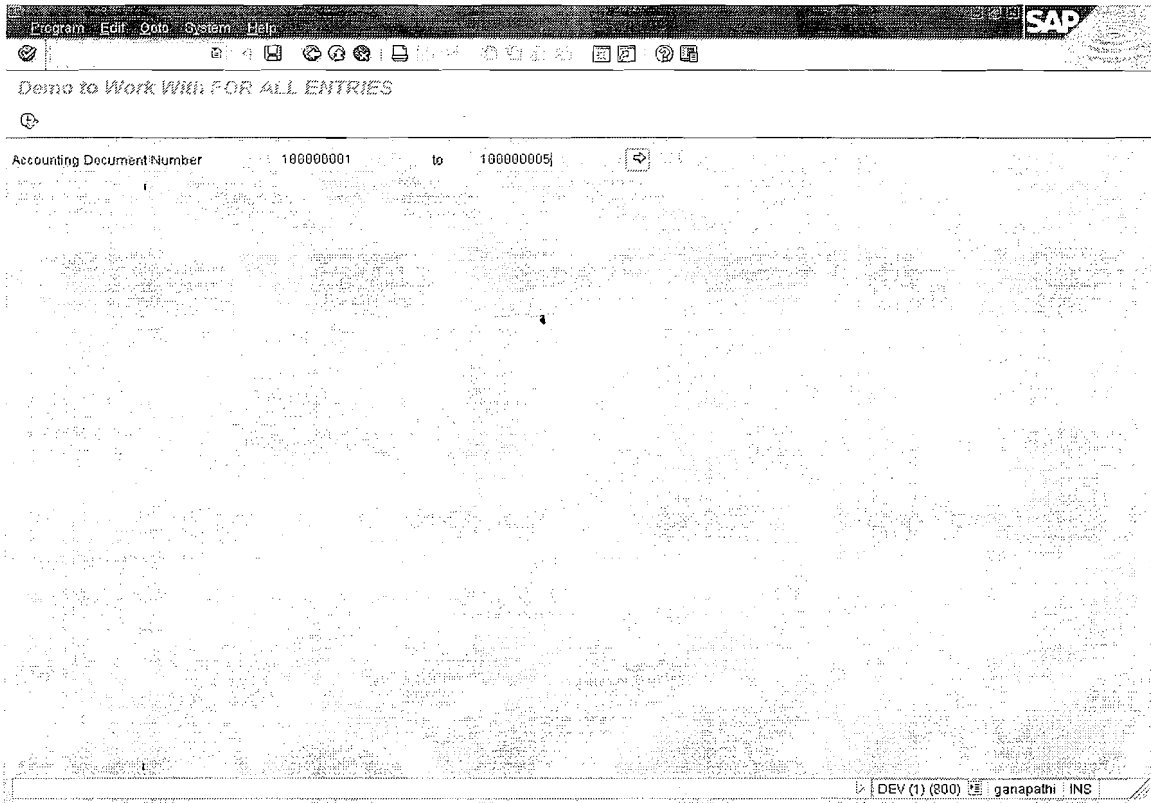
```
WRITE :/' CODE','ACC DOC NO',
'REF DOC NO','POSTING DATE','ITEM NO',
'BUSINESS AREA','ASSIGNMENT NO','AMT IN CURR','CUST NO'.
```

```
ULINE.
IF SY-SUBRC = 0.
```

```
LOOP AT IT_FINAL INTO WA_FINAL.
WRITE :/1 SY-VLINE,WA_FINAL-BUKRS UNDER 'COM CODE',SY-VLINE,
      WA_FINAL-BELNR UNDER 'ACC DOC NO',SY-VLINE,
      WA_FINAL-XBLNR UNDER 'REF DOC NO',SY-VLINE,
```

```
WA_FINAL-BUDAT UNDER 'POSTING DATE',SY-VLINE,  
WA_FINAL-BUZEI UNDER 'LINE ITEM NUMBER',SY-VLINE,  
WA_FINAL-GSBER UNDER 'BUSINESS AREA',SY-VLINE,  
WA_FINAL-ZUONR UNDER 'ASSIGNMENT NO',SY-VLINE,  
WA_FINAL-WRBTR UNDER 'AMT IN CURR' RIGHT-JUSTIFIED,SY-VLINE,  
WA_FINAL-KUNNR UNDER 'CUST NO',SY-VLINE.  
ENDLOOP.  
ELSE.  
WRITE :/ 'NO record Found'.  
  
ENDIF.
```

OUTPUT:- EXECUTE THE PROGRAM



List Edit Goto System Help SAP

DEMO TO WORK WITH FOR ALL ENTRIES

DEMO TO WORK WITH FOR ALL ENTRIES

CODE	ACC	DOC	NO	REF	DOC	NO	POSTING	DATE	ITER	NO	BUSINESS	AREA	ASSIGNMENT	NO	AMT	IN	CURR	CUST	NO
1000	00000001	11583794	09	11	1994	001	1000								212.173				
1000	00000001	11583794	09	11	1994	002	1000					11583794			244.800				
1000	00000001	11583794	09	11	1994	003	1000								31.925				
1000	00000001	11583794	09	11	1994	001	9100					19991205			275				
1000	00000001	11583794	09	11	1994	002	9100								275				
1000	00000001	11583794	09	11	1994	001						0000051207917			512.079				
1000	00000001	11583794	09	11	1994	002						20019305			512.079				
2000	00000001	18 05 1995	001	9900	2-1000										2.830				
2000	00000001	18 05 1995	002	9900	2-1000										1.810				
2000	00000001	18 05 1995	003	9900	2-1000										2.280				
2000	00000001	18 05 1995	004	9900	2-1000										4.470				
2000	00000001	18 05 1995	005	9900	2-1000										3.560				
2000	00000001	18 05 1995	006	9900	2-1000										980				
2000	00000001	18 05 1995	007	9900	19950518										15.330				
2000	00000001	18 05 1995	001		20001231										3.364				
2000	00000001	18 05 1995	002		20001231										3.364				
2000	00000001	18 05 1995	001	9900	2-1000										26.387				
2000	00000001	18 05 1995	002	9900	000000										26.407				
2100	00000001	0000000002	27	05	1995	001									670.80026601				
2100	00000001	0000000002	27	05	1995	002						19950827			670.800				
2100	00000001	0000000002	27	05	1995	001	2000								1.945				
2100	00000001	0000000002	27	05	1995	002	2000								1.945				
2100	00000001	0000000002	27	05	1995	003	2000								57				
2100	00000001	0000000002	27	05	1995	004	9000								57				
2200	00000001	TEST INTERC28	08	1997	001	7220						0000003575			83.8172500				
2200	00000001	TEST INTERC28	08	1997	002	7220									89.590				
2200	00000001	TEST INTERC28	08	1997	003	7220						19870828			14.317				
2200	00000001	TEST INTERC28	08	1997	001	7220						20001231			69.754				
2200	00000001	TEST INTERC28	08	1997	002	7220						20001231			69.754				
2200	00000001	TEST INTERC28	08	1997	003	7220						20001231			4.219				
2200	00000001	TEST INTERC28	08	1997	004	7220						20001231			4.213				
2400	00000001	AP0	08	09	2000	001	1000					0000007587			156.8802401				
2400	00000001	AP0	08	09	2000	002	1000					20000908			150.880				

DEV (1) (800) ganapathi INS

THINGS TO BE CONSIDERED WHEN WORKING WITH FOR ALL ENTRIES:

"For all entries in" 3 pitfalls :

```
select shkzg wrbtr saknr "debit/credit indicator, amount, GL acct
from bseg into table t_bseg
for all entries in it_bkpf
where belnr = it_bkpf-belnr and bukrs = 'CUR '.
```

This is the equivalent of saying "select distinct shkzg wrbtr saknr"

Duplicates are removed from the answer set as if you had specified "SELECT DISTINCT"... So unless you intend for duplicates to be deleted **include the unique key** of the detail line items in your select statement. It will only pick up one line item if multiple line items appear with the same debit/credit indicator, amount and GL Account. **If you want all occurrences of these you must have a select statement that includes the table's unique key, also called primary key.**

Instead Do:

```
SELECT bukrs belnr gjahr buzei shkzg wrbtr saknr "bseg unique key + d/c ind,
amt, GL acct from BSEG into table t_bseg for all entries in t_bkpf where belnr =
t_bkpf-belnr and bukrs = 'CUR '.
```

FOR ALL ENTRIES IN...acts like a range table, so that if the "one" table is empty, all rows in the "many" table are selected. Therefore make sure you check that the "one" table has rows before issuing a select with the "FOR ALL ENTRIES IN..." clause.

IF NOT IT_BKPF IS INITIAL.

```
SELECT BUKRS BELNR GJAHR BUZEI SHKZG
WRBTR SAKNR
"BSEG UNIQUE KEY + d/c ind, amt, GL acct
FROM BSEG INTO TABLE IT_BSEG.
FOR ALL ENTRIES IN IT_BKPF
where belnr = it_bkpf-belnr and bukrs = 'CUR '.
```

ENDIF. "if there are any projects

Note:So that having the **IF NOT IT_BKPF IS INITIAL. Is Mandatory.**

NOTE : If the parent table (it_bkpf) is very large there is performance degradation

WORKING WITH SELECT SINGLE & UP TO 1 ROWS :

Note: Both retrieves Only One Record Always.

	SELECT SINGLE	SELECT UP TO 1 ROWS
Syntax	SELECT <List Of Fields> into <WA> FROM <DBT> WHERE <Condition>.	SELECT <List Of Fields> into <WA> FROM <DBT> UP TO 1 ROWS WHERE <Condition>.
1	It Reads the Exact Record	It Reads the First Record Found(May not be Exact)
2	The WHERE Condition Should Include all the Primary Keys To Identify the record Uniquely.	The WHERE Condition Can have the Part of the Primary Key.
3	Prefer this to read the Exact Information.	Prefer this to Validate Input as we are not Interested with the Exact Match. Because a record is Valid if at least one record found.

Note1 :

When we Don't Pass all the Primary Keys in the SELECT SINGLE , it Acts as SELECT UP TO 1 ROWS, but doesn't throw error.

Note 2 : If the Input is SELECT-OPTIONS We cannot expect SINGLE Record and where as we can expect UP TO 1 ROWS because it can be any no of records for the Input Range.

REPORT ZDEMO_SELECT_SINGLE_UPTO_1 MESSAGE-ID ZDEMO.

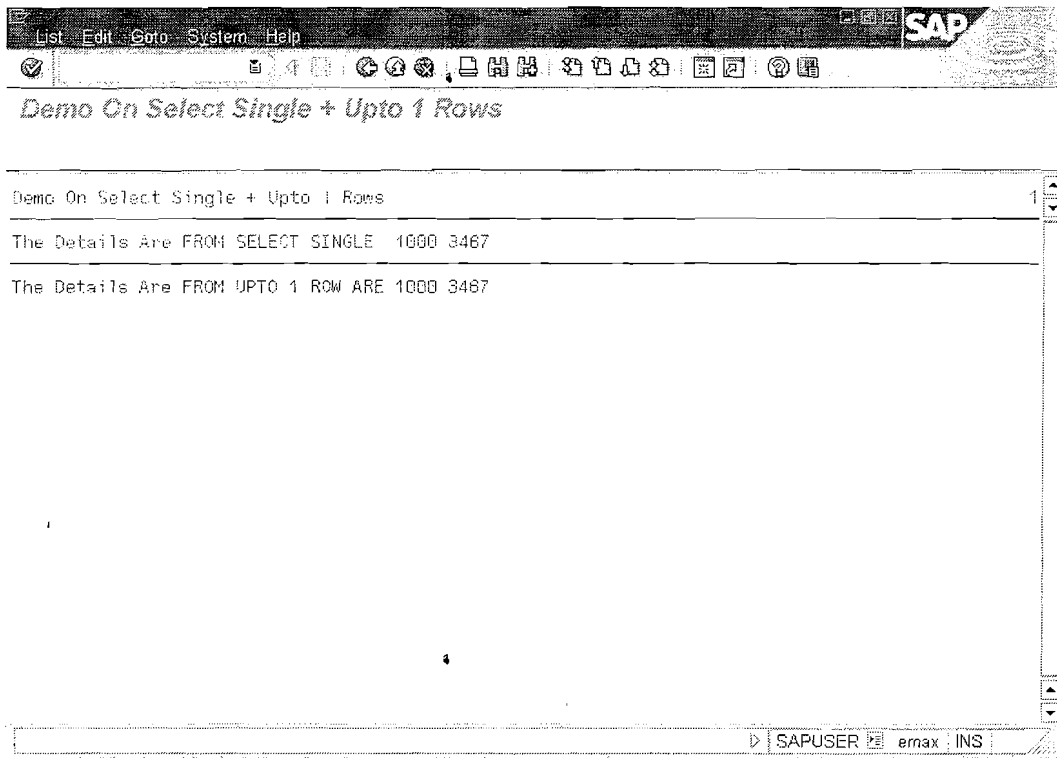
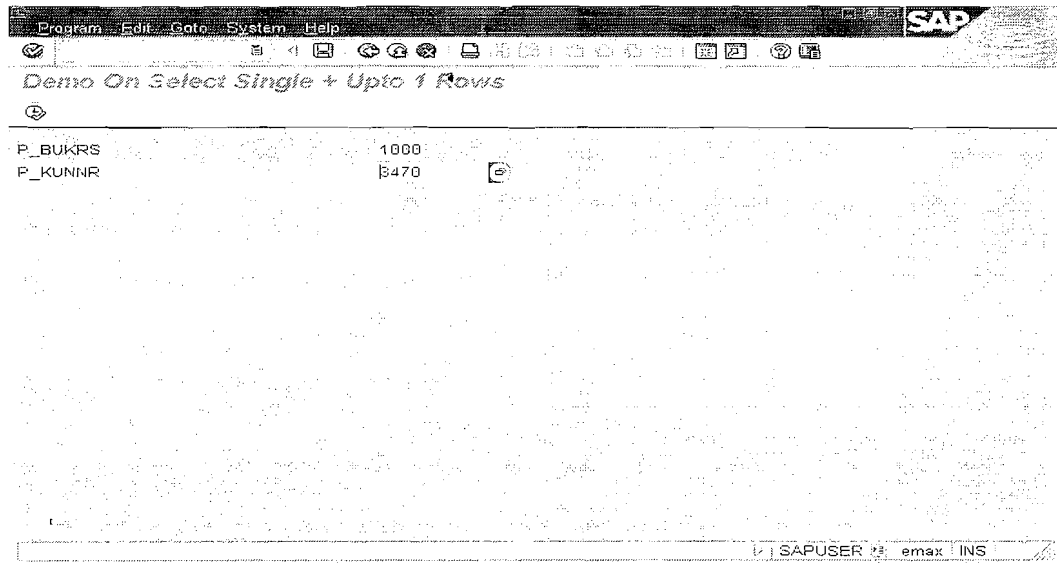
```
*****
* PROGRAM       : ZDEMO_SELECT_SINGLE_UPTO_1
* AUTHOR        : GANAPATHI.ADIMULAM
* START DATE    : 27/01/2008
* PURPOSE       : IS TO WORK WITH SELECT SINGLE AND
                  SELECT UPTO 1 ROWS
* COPIED FROM   : NA.
*****
* MODIFICATION LOG:
* CHANGE REQUEST : C11EMAX4756
* -----
*MOD-001       : DETAILS OF MODIFICATION 001
*MOD-002       : DETAILS OF MODIFICATION 002
*SUPPLIER      : EMAX TECHNOLOGIES.AMEERPET
*****
```

DATA : V_BUKRS TYPE BUKRS, " COMPANY CODE
V_KUNNR TYPE KUNNR. " CUSTOMER NUMBER

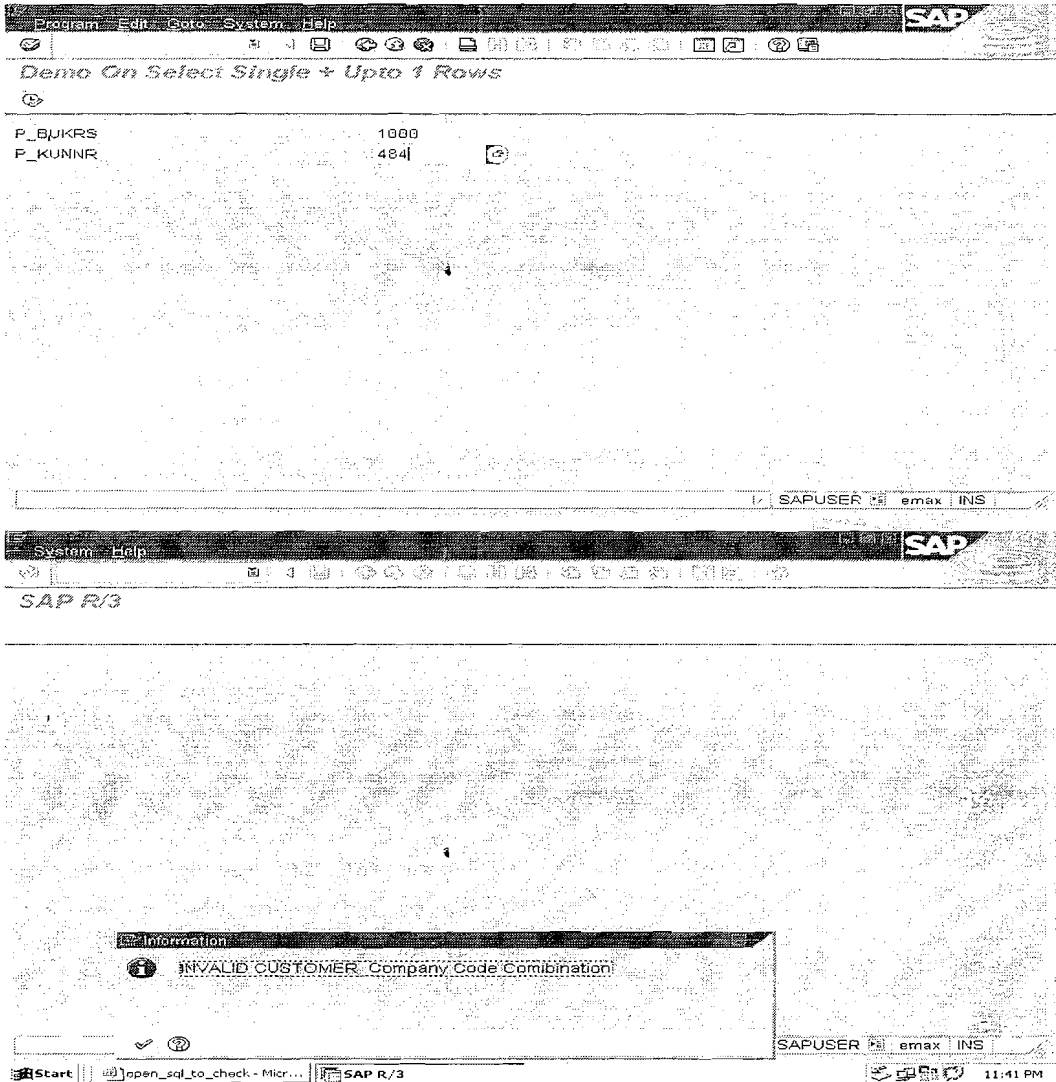
```
PARAMETER : P_BUKRS TYPE BUKRS,
            P_KUNNR TYPE KUNNR.
SELECT SINGLE BUKRS KUNNR INTO (V_BUKRS , V_KUNNR)
FROM KNB1
WHERE BUKRS = P_BUKRS AND
      KUNNR = P_KUNNR.
IF SY-SUBRC = 0.
  WRITE : / 'The Details Are' , V_BUKRS, V_KUNNR.
ELSE.
  MESSAGE I000.
ENDIF.
CLEAR : V_BUKRS,V_KUNNR.
SELECT BUKRS KUNNR INTO (V_BUKRS , V_KUNNR)
UP TO 1 ROWS
FROM KNB1
WHERE KUNNR = P_KUNNR.
```

```
ENDSELECT.  
ULINE.  
IF SY-SUBRC = 0.  
  WRITE : / 'The Details Are', V_BUKRS,V_KUNNR.  
ELSE.  
  WRITE : / ' It is Not a Valid Customer'.  
ENDIF.
```

OUTPUT :- EXECUTE THE PROGRAM



In case of invalid company code we get the following screen.



DATA CONTROL LANGUAGE

Committing Database Changes

COMMIT WORK. (To Save the Changes Permanently in DB)

ROLLBACK WORK to Undo the Changes Which are not Yet Committed.

Exercises

1. List all the rows from the table LAGP (Storage Bins)
2. List all the rows from the table EINA (Purchase Info Record) for the given Range of Vendor.
3. List up to 5 rows from table EBAN (Purchase Requisition)
4. List all the rows from the tables KNA1 (Customer General Data), KNB4 (Customer Payment History) for the given range of customers.
5. Accept Doc.No. From user, display doc.no. doc.status, date of docu. Docu.type and Item Details from BKPF and BSEG.
7. Accept document no. from user and display the particulars of the Sales Docu. No, corresponding material no. description of that material and item category. (Table VBAP, default docu. no. '0010000031').



8. ABAP Debugger

Duration in Days - 1(* 2 Hrs)

- a. Purpose**
- a. Features**
- b. Starting the Debugger**
- c. Break Points, Types and Setting Break Points**
- d. Display Modes in Debugging**
- e. Watch Points**
- f. Execution Types in Debugging Mode**

ABAP Debugger:**Use**

The ABAP Debugger is an integrated test tool within the ABAP Workbench. You use it to trace the source code of an ABAP program. In the Debugger, you can step through the source code of a program. The running program can be interrupted after each step, allowing us to check its processing logic and the results of individual statements.

Features

The Debugger provides an efficient means of identifying errors in ABAP programs.

It contains the following functions:

- Ways of starting the Debugger
- Choosing Debugger settings
- Choosing different execution options in the Debugger
- Displaying source code in the Debugger
- Setting and deleting breakpoints
- Setting and deleting watchpoints
- Stopping a program at a particular statement, event, subroutine, or function module
- Displaying and changing field contents at runtime

Starting the Debugger

- **By setting breakpoints then running the program**
- **By running the program in debugging mode.**

Setting Breakpoints :

A breakpoint is a signal in a line of code that tells the ABAP runtime processor to interrupt the program at that line and start the ABAP Debugger. **Breakpoints are useful when you want to analyze a program further in the following cases:**

- After the system has processed particular events,
- Before a particular event is triggered,
- When you want to go directly to a particular routine or call.

Running a Program in Debugging Mode

You can start the Debugger without previously having set breakpoints. **This is the best procedure to use when you want to test a program right from the beginning.** You can also use it to debug a transaction starting with the first PBO module. It is also useful if you do not know the program very well and therefore are not sure where best to set breakpoints.

You can start the Debugger as follows:


From the Object Navigator	Select a program and choose <i>Test/Execute</i> from the <i>Development Object</i> menu. The <i>Choose Execution Type</i> dialog box appears. Choose <i>Debugging</i>
From the initial screen of the ABAP Editor	Choose Program → Execute → <i>Debugging</i> (or the <i>Debugging</i> pushbutton).
From any screen	Choose <i>System</i> → <i>Utilities</i> → <i>Debug ABAP</i> .
From any screen	Enter "/h " in the command field(Transaction Bar).



Display Modes in the Debugger

When you are debugging a program, there are various display modes that you can use. The Goto menu allows you to switch between the different modes. **There are also pushbuttons on the screen allowing you to switch to the most frequently-used.**

Display Modes Available Using Pushbuttons

Fields	Table	Breakpoints	Watchpoints	Calls	Overview	Settings
--------	-------	-------------	-------------	-------	----------	----------

<p>Fields</p>	<p>The scrollable field display contains the contents of up to eight fields. The contents of the three most important system fields are always displayed. This is the default display mode in the Debugger. See also:</p> <p>Processing Fields : Enter a field name directly in one of the fields in the display, or select a field by double-clicking its name in the source code display. When you select a field from the source code display, the system automatically enters it in the field display.</p> <p>This display mode allows you to display the contents of any fields defined in the program, but also any system fields. The three most important system fields (SY-SUBRC, SY-TABIX, and SY-DBCNT) are always displayed at the bottom of the screen.</p> <p>Deleting All Field Names</p> <p>Use this button  to delete all field names from the field display.</p>
<p>Table</p>	<p>Displays the contents of an internal table. This mode allows you to display and edit the entries in an internal table.</p> <p>To Display Internal Table Contents:</p> <ol style="list-style-type: none"> 1. The <i>Table</i> pushbutton from any other display mode. 2. Enter the name of an internal table in the <i>Internal table</i> field, or select <itab> by double-clicking its name in the source code. <p>We Can Append, Delete, Modify Internal Table Contents. Select the</p>

	Record and click on the Relevant Button. Then Enter the new Details and Press ENTER .
Breakpoints	A scrollable display containing up to 30 breakpoints . Next to each breakpoint is a counter. You can also delete breakpoints in this display.
Watchpoints	Which are Intelligent Break Points. Through Which the Program Can be Interrupted by checking the Contents Field. You can set a watchpoint for a field so that the program is interrupted whenever the value of that field changes.
Calls	This mode displays the current sequence of events, and the sequence of calls up to the current breakpoint. The last active call is displayed at the top of the list; previous calls are listed in reverse chronological order. When an event (for example, START-OF-SELECTION) concludes, it is deleted from the display.
Overview 	This mode displays the structure of the program. It lists its events, subroutines, and modules, and shows which sections belong to which events. It also displays the section currently being processed.
Settings 	This mode displays the current Debugger settings. You can change the settings by selecting or deselecting various options. For further information, refer to:

Breakpoints :

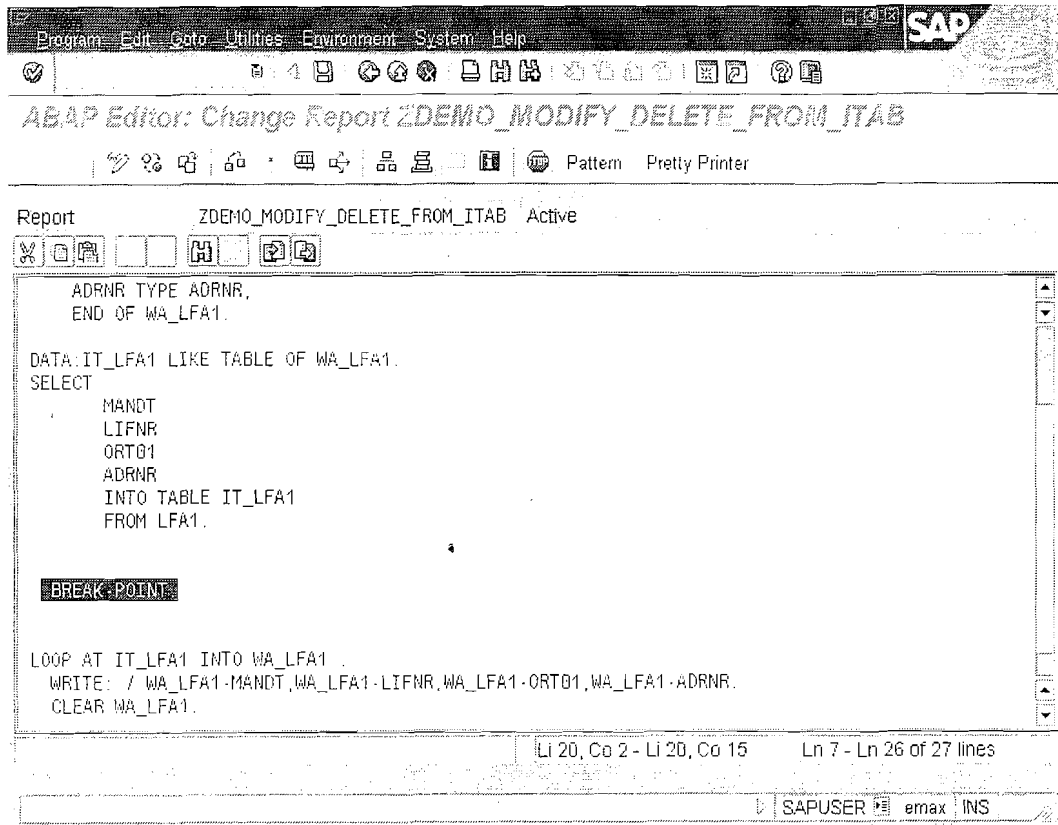
Instead of starting an ABAP program directly in the Debugger, you can also debug a program by creating one or more breakpoints in the program. A **breakpoint is a signal at a particular point in the program that tells the ABAP runtime processor to interrupt processing and start the Debugger. The program runs normally until the breakpoint is reached.**

Different Types Of Breakpoints :

Static	The BREAK-POINT statement in an ABAP program. Static breakpoints are not normally user-specific. However, you can make them user-specific.
Directly-set dynamic breakpoints	Can be set in the ABAP Editor or the Debugger. Dynamic breakpoints are displayed as stop signs in the Debugger and ABAP Editor. Unlike static breakpoints, they are user-specific , and are deleted when you log off from the R/3 System.

Static Breakpoints :

Static breakpoints are **not normally user-specific**. Once a user has inserted a **BREAK-POINT** statement in an ABAP program, the system always interrupts the program at that point. **You should set static breakpoints whenever more than one programmer is working on the same program and you always want to stop the program in the same place.** You should only use static breakpoints during the development phase of an application.

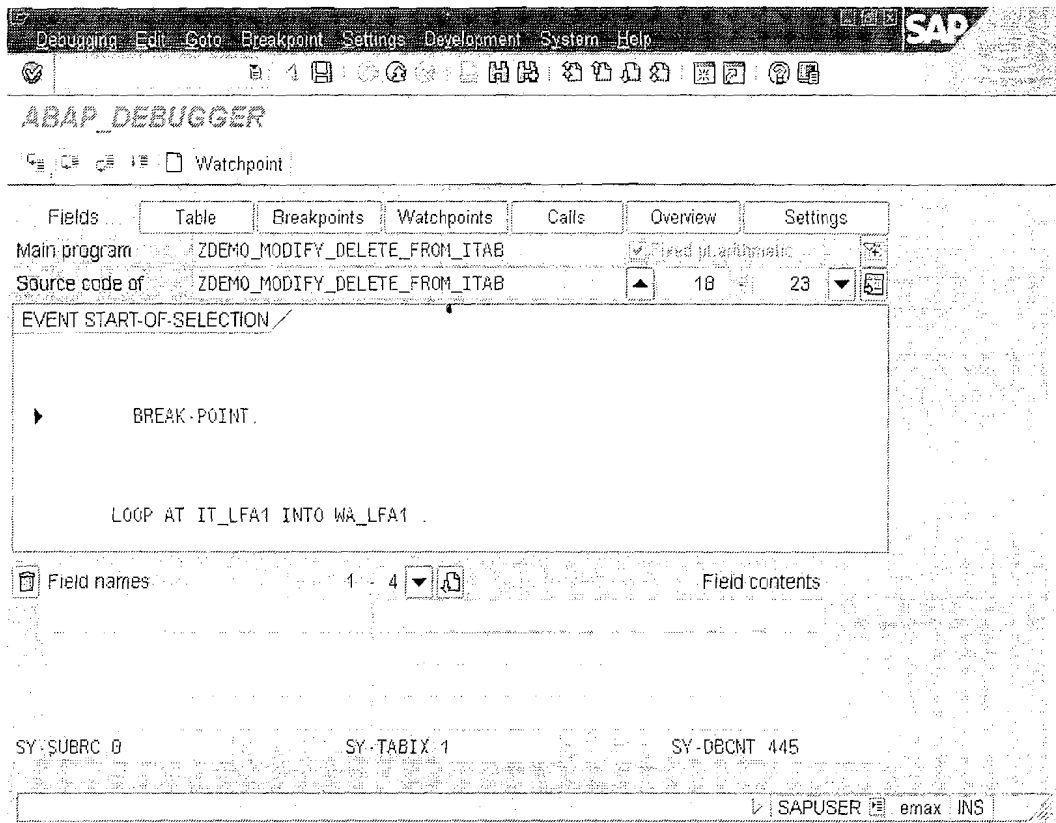


The screenshot shows the SAP ABAP Editor interface. The title bar reads "ABAP Editor: Change Report ZDEMO_MODIFY_DELETE_FROM_ITAB". The menu bar includes "Program", "Edit", "Goto", "Utilities", "Environment", "System", and "Help". The status bar at the top right shows "SAP". The main editor area displays the following ABAP code:

```
ADRNR TYPE ADRNR,  
END OF WA_LFA1.  
  
DATA:IT_LFA1 LIKE TABLE OF WA_LFA1.  
SELECT  
  MANDT  
  LIFNR  
  ORTB1  
  ADRNR  
  INTO TABLE IT_LFA1  
  FROM LFA1.  
  
BREAK-POINT  
  
LOOP AT IT_LFA1 INTO WA_LFA1 .  
  WRITE: / WA_LFA1-MANDT,WA_LFA1-LIFNR,WA_LFA1-ORTB1,WA_LFA1-ADRNR.  
  CLEAR WA_LFA1.
```

At the bottom of the editor, the status bar shows "LI 20, Co 2 - Li 20, Co 15" and "Ln 7 - Ln 26 of 27 lines". The user information bar at the very bottom shows "SAPUSER", "emax", and "INS".

This Program will be always executed for all the Users because of the Statement **BREAK-POINT** in the Source Code.



Observe it is Stopped for the Execution of the Program. So it is always better to avoid Static Break points.

Dynamic Breakpoints

Dynamic breakpoints are **user-specific**. You should therefore use them when you only want the program to be interrupted when you run it yourself. **All dynamic breakpoints are deleted when you log off from the R/3 System.**

Dynamic breakpoints are more flexible than static breakpoints, because you can deactivate or delete them at runtime.

Special Dynamic Breakpoints

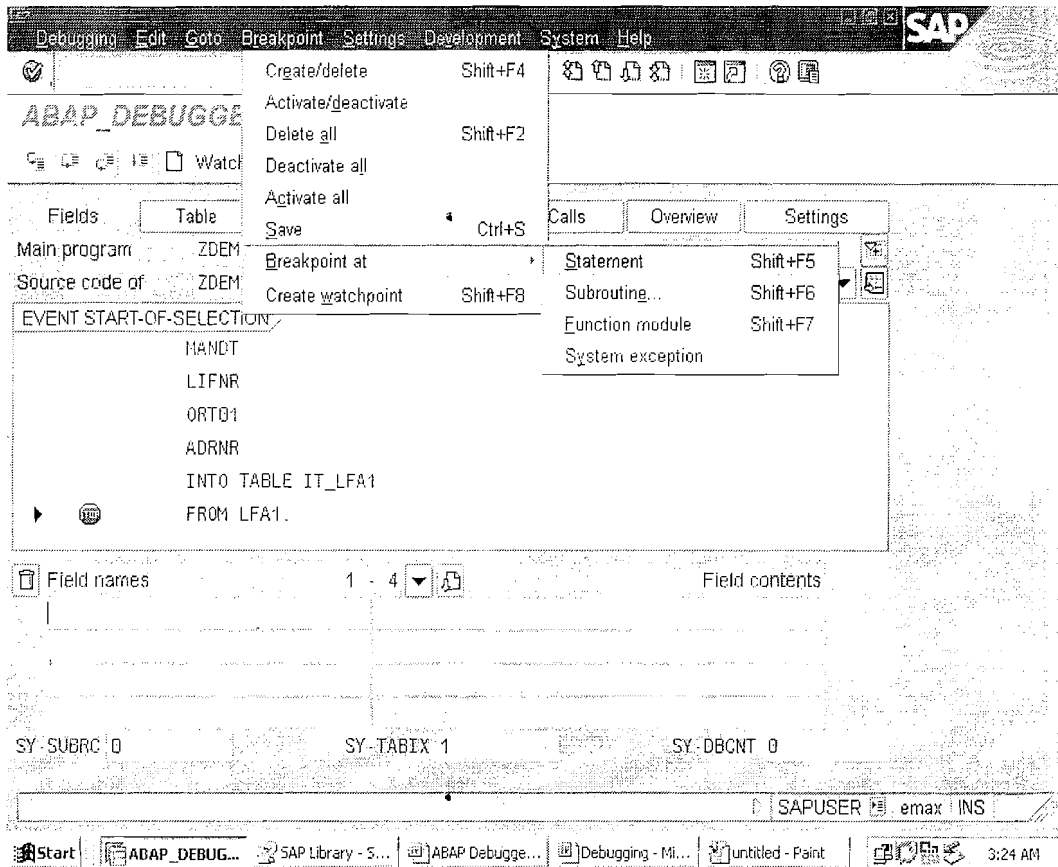
Special dynamic breakpoints are useful when you want to interrupt a program directly before a particular ABAP statement, a subroutine, or an event, but do not know exactly where to find it in the program code. Special dynamic breakpoints are user-specific. You can only set them in the Debugger.

Breakpoints at Statements

Breakpoints at Subroutines

Breakpoints at Function Module Calls

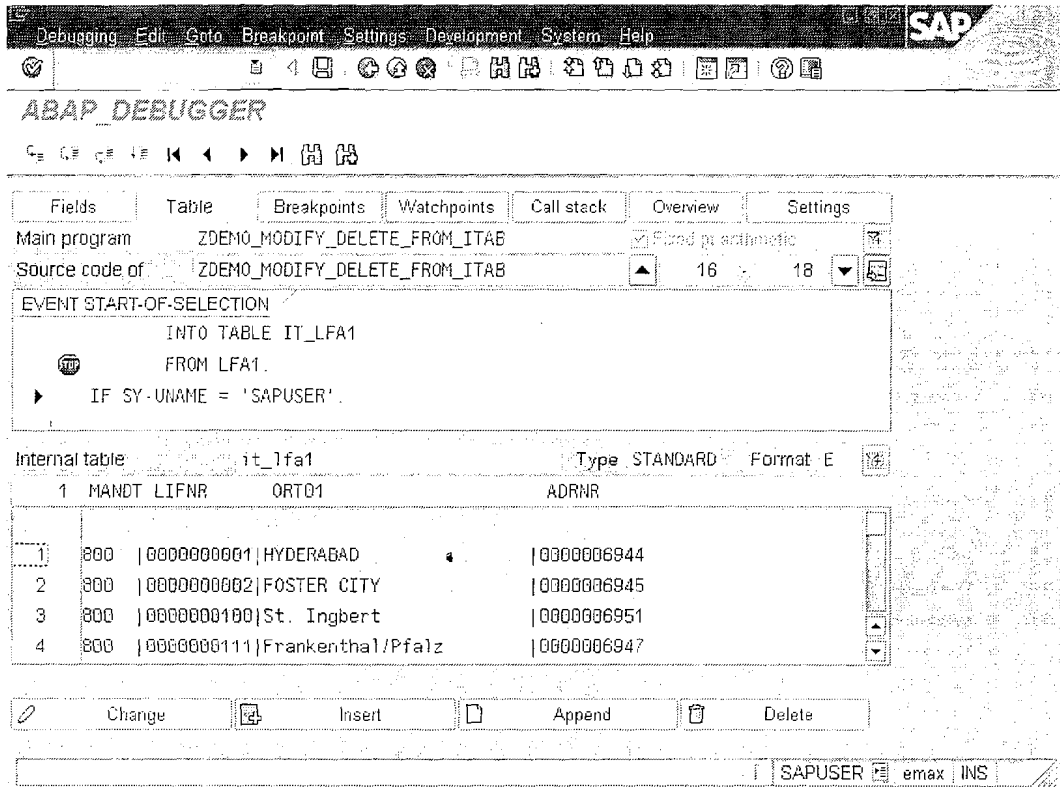
Breakpoints at System Exceptions



Operations On Internal table in Debugging Mode

Click on Table tab and enter the <ITAB> name





Here you Select the Record, Click On Change/Insert/Delete for the respective Operations.

Steps to Create/Set Watchpoints:

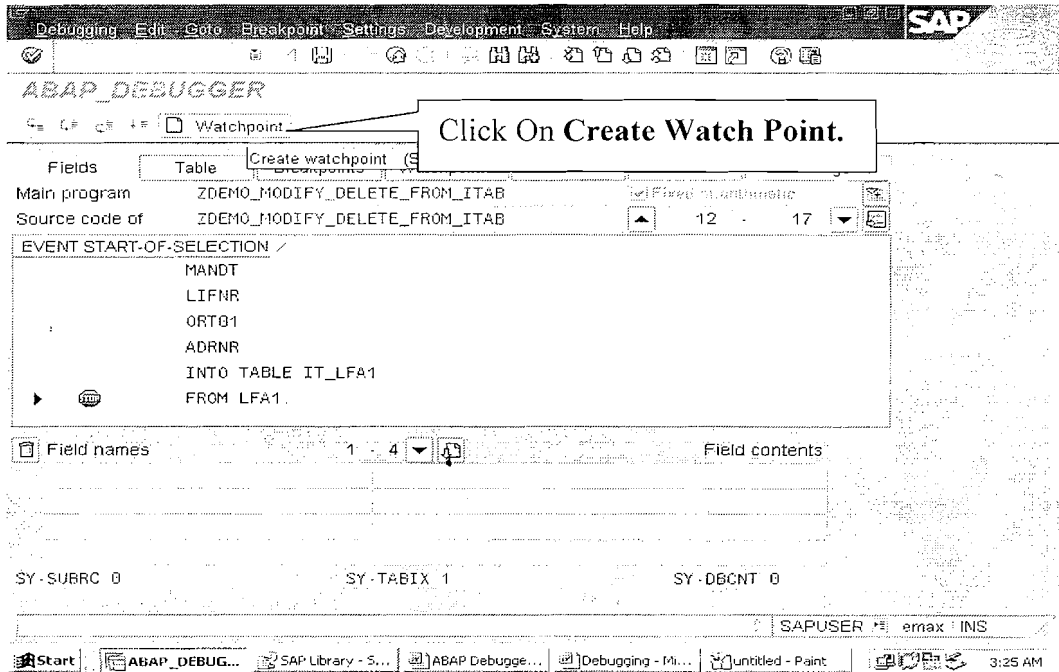
If you want to interrupt a program when the contents of a field or structure change, use a watchpoint. You can set up to five watchpoints, including watchpoints for **strings**.

A watchpoint may be either **local** or **global**: Local watchpoints are only valid in the specified program. Global watchpoints are valid in the specified program, and also in all other programs that it calls.

Procedure

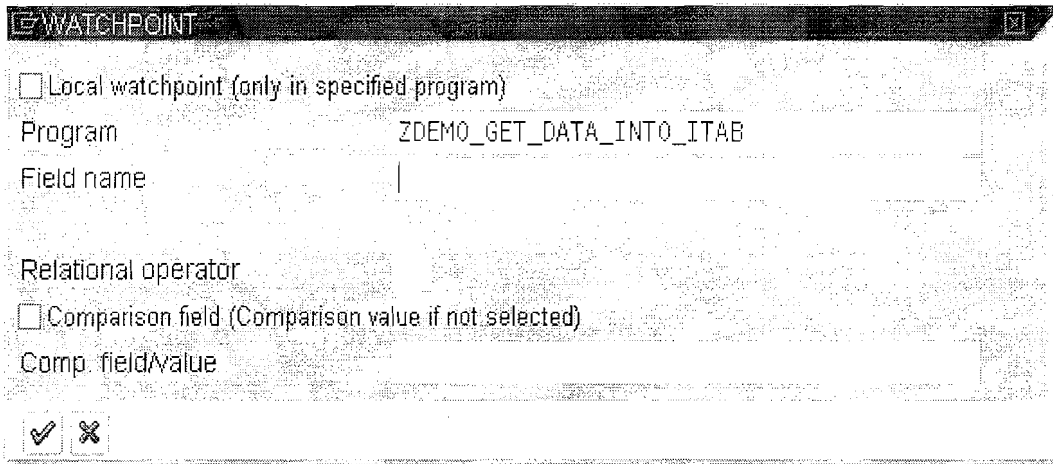
To set a watchpoint, start the Debugger and proceed as follows:

1. Choose *Breakpoint* → *Create watchpoint* or the corresponding pushbutton.



The *Create Watchpoint* dialog box appears:

2. Decide whether you want to set a local or global watchpoint.
3. Enter the name of the field for which you want to set the watchpoint.
4. If you want your watchpoint to be activated each time the contents of the field change, the definition is now complete, and you can return to the Debugger by pressing **ENTER**. The watchpoint appears in the watchpoint display. If you want to specify conditions on which the watchpoint should be activated, continue with step 6.



5. To create a conditional watchpoint, choose a relational operator. Valid operators:

Operator	Meaning
----------	---------

= or EQ	Equal
<> or NE	Not equal
< or LT	Less than
<= or LE	Less than or equal
>= or GE	Greater than or equal
> or GT	Greater than

6. You can use the *Comparison field* option to specify whether the comparison should be with a value that you specify or with the contents of another field.

7. Depending on your choice from step 6, enter a value or a field for the comparison.

WATCHPOINT

Local watchpoint (only in specified program)

Program: ZDEMO_GET_DATA_INTO_ITAB

Field name: wa_t001-bukrs

Relational operator: =

Comparison field (Comparison value if not selected)

Comp. field/value: 3000

8. Choose **ENTER** to create the conditional watchpoint.
Execute it (F8).

The image contains two screenshots of the SAP ABAP Debugger interface. The top screenshot shows the 'Watchpoint' tab with a watchpoint set on the line 'WRITE : /S WA_T001-BUKRS.'. A callout box points to the watchpoint icon with the text 'Observe Watch Point is Reached.' The bottom screenshot shows the same interface after the watchpoint has been reached. A callout box points to the 'Field contents' section, which displays '3000IDES-USA INC' and 'New York'. The text 'Observe the Contents of BUKRS i.e 3000. Which Was Already defined in WatchPoint.' is overlaid on this section.

Stepping Through the Source Code

There are four different ways in which you can step through the source code of a program you want to analyze





F5 - Single Step

F6 - Execute the Subroutine/Function Module ... at a time with out entering into the Definition.

F7 - After Entering the Definition of The FORM –ENDFORM/FUNCTION- ENDFUNCTION, To Come Out from any Point of Definition.

F8 - Jumps to Next Break Point/Watch Point if available .Otherwise It Executes the Whole Program and Comes Out Of it.

Execution Types in Debugging Mode:

<p><i>Single step</i> —  F5</p>	<p>Use this option to step through the program statement by statement. This allows you to branch into subroutines and function modules, and to execute these routines step by step as well. Once a subroutine or function module has been processed, control returns to the statement following the CALL FUNCTION or PERFORM statement.</p>
<p><i>Execute</i> — F6 </p>	<p>Use this option to process a program line by line. All of the statements on the current line are processed in a single step. If you are positioned on a line that calls a subroutine and you choose <i>Execute</i>, the Debugger processes the whole subroutine and then moves on to the line following the subroutine call. This allows you to jump through the statements within the subroutine.</p>
<p><i>Return</i> — F7 </p>	<p>The Debugger returns from a routine to the point at which control returns to the main program. Use this option to return from a subroutine, function module, or called program to the calling program.</p>
<p><i>Continue</i> — F8 </p>	<p>Use this option to process the program up to the next dynamic or static breakpoint or up to the cursor position. If there are no more breakpoints in the program and no cursor has been set, the system exits debugging mode and executes the rest of the program normally.</p>

9.REPORTS

Duration in Days - 5(* 2 Hrs)

- a. Classical Reports
- b. Defining selection Screen
 - i. PARAMETER
 - ii. SELECT-OPTIONS
 - iii. Check Boxes, Radio Buttons
- c. Formatting Selection Screens
 - i. Blank Lines, Underlines, Comments, Blocks

D. Classical Report Events:

- a. INITIALIZATION.
- b. AT SELECTION-SCREEN.
- c. AT SELECTION-SCREEN ON.
- d. START-OF-SELECTION.
- e. TOP-OF-PAGE.
- f. END-OF-PAGE.
- g. END-OF-SELECTION.

e. Interactive Report Events:

- AT LINE-SELECTION.
- AT USER-COMMAND.
- TOP-OF-PAGE DURING LINE-SELECTION.

f. System Fields for Detail Lists

Passing Data by Program Statements

- .HIDE
- .GET CURSOR
- .READ LINE

Reports

We Never Compromise In Quality, Would You?

DAY-1

Introduction :

Report is displaying the application data in the required format. Technically speaking, a report is an executable program with three stage function:

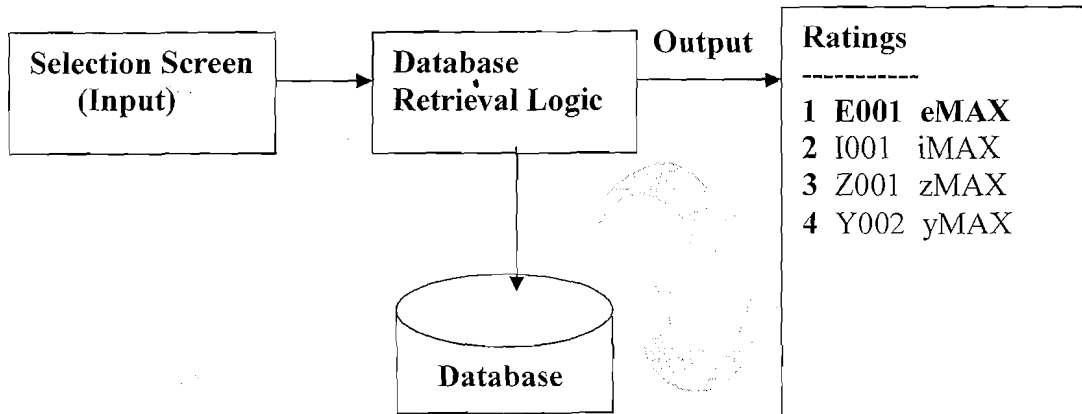
Data input -> data processing -> data output.

DATAINPUT (Selection Screen)

DATA PROCESSING (SELECT Statements)

DATAOUTPUT. (Write, Skip, Uline, Vline etc to Output the Data).

Ratings Of SAP Institutes:



Purpose : It helps to analyze the current situation and also for decision Making.

I.e. to Decide the right Institute using the above Report.

Real time Report Scenarios:

i.e. to stop the Purchases from the Specific Vendors, the Purchasing Department need a report for **Vendor Evaluation** and to terminate the employees from the Organization ,the HR Department need one report with **Employee Performance etc.**

Note: Reports read and calculate data from **database tables**, without actually changing it.

Reports are of two Types :

a) Classical Reports

Definition: Displaying the whole data as One List.

b) Interactive Reports

Definition: Display the Summarized Information as the First List

And Display the detailed information as Secondary Lists.

Working With CLASSICAL Reports

Note : Either Classical and Interactive Reports , Providing the INPUT is always through SELECTION SCREEN Only.

Defining Selection Screens :

ABAP programs use Selection screens to obtain input from users.

There are three ABAP statements for defining selection screens:

PARAMETERS	for single fields
SELECT-OPTIONS	For Single and Multiple Range Of Fields
SELECTION-SCREEN	for formatting the selection screen and defining user-specific selection screens

Standard selection screens

The standard selection screen of executable programs is predefined and has screen number **1000**.

PARAMETER in Detail : Each parameter declared with the PARAMETER(S) statement appears as an input field on the relevant selection screen.

Parameters are used for simple queries of single values ONLY.

SYNTAX For Parameters :

PARAMETER <Name>[(<length>)] TYPE <type>| LIKE <Variable>
[DECIMALS <No>].

Note: Currently, the Length Of parameter names are limited to eight .

Note: The data types valid for parameters include all elementary ABAP types except **data type F**.

If the parameter refers to data types from the Dictionary, it adopts all attributes of the Dictionary field. Currently, parameters can only refer to fields of database tables, views and structures. In particular, the field help (F1) and the possible entries help (F4) defined for these fields in the Dictionary are available to the user.

Default Values for Parameters :

PARAMETERS p_bukrs TYPE BUKRS DEFAULT '1000' .

The input field of the parameter on the selection screen is filled with the default value. The user can accept or change this value.

Defining PARAMETER as Mandatory (Required) Fields :

PARAMETER p_bukrs TYPE BUKRS **OBLIGATORY**.

Note : The user cannot continue with the program without entering a value in this field on the selection screen.

Checking Input Values :

To check a user entry against a **check table or against fixed values** in the ABAP Dictionary.

PARAMETER <Name> TYPE <Data type> **VALUE CHECK** .

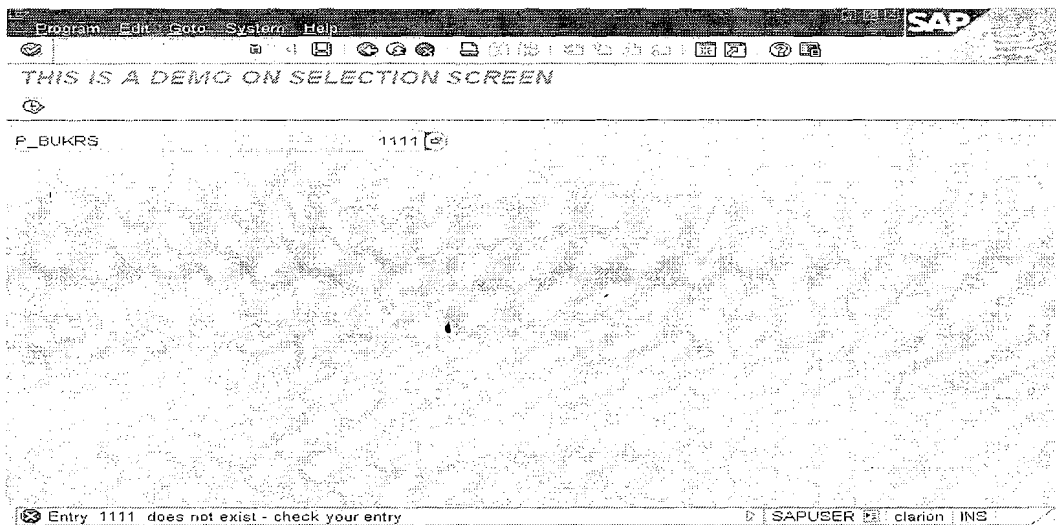
PARAMETERS P_BUKRS LIKE KNB1-BUKRS OBLIGATORY VALUE CHECK.

Parameter P_BUKRS is declared with reference to field BUKRS of database table KNB1.

For this field, check table T001 is specified in the ABAP Dictionary.

The user can only enter Company Code values that are contained in T001.

The possible entries help of the input field for P_BUKRS displays the allowed values.



Note: Since Company code '1111' doesn't exist in T001.

Defining Checkboxes:

PARAMETER <Name> AS CHECKBOX.

Parameter <Name> is created with type C and length 1.

Defining Radio Buttons:

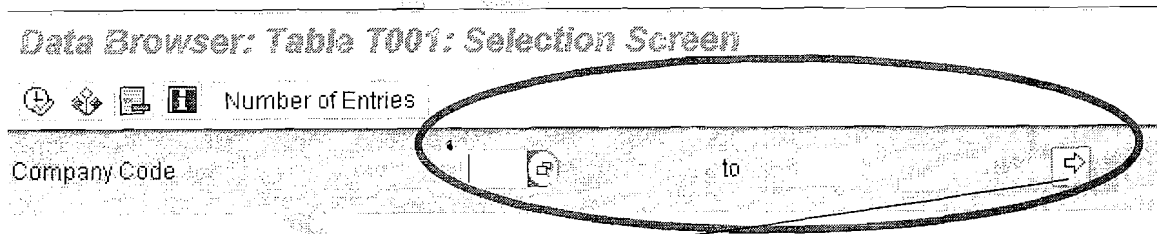
PARAMETER <Name> RADIOBUTTON GROUP <G1 >.

Parameter <Name> is created with type C and length 1, and is assigned to **group** <G1>. Each Radiobutton Group has atleast Two Radiobuttons and by default the First Radiobutton from the group is Selected .

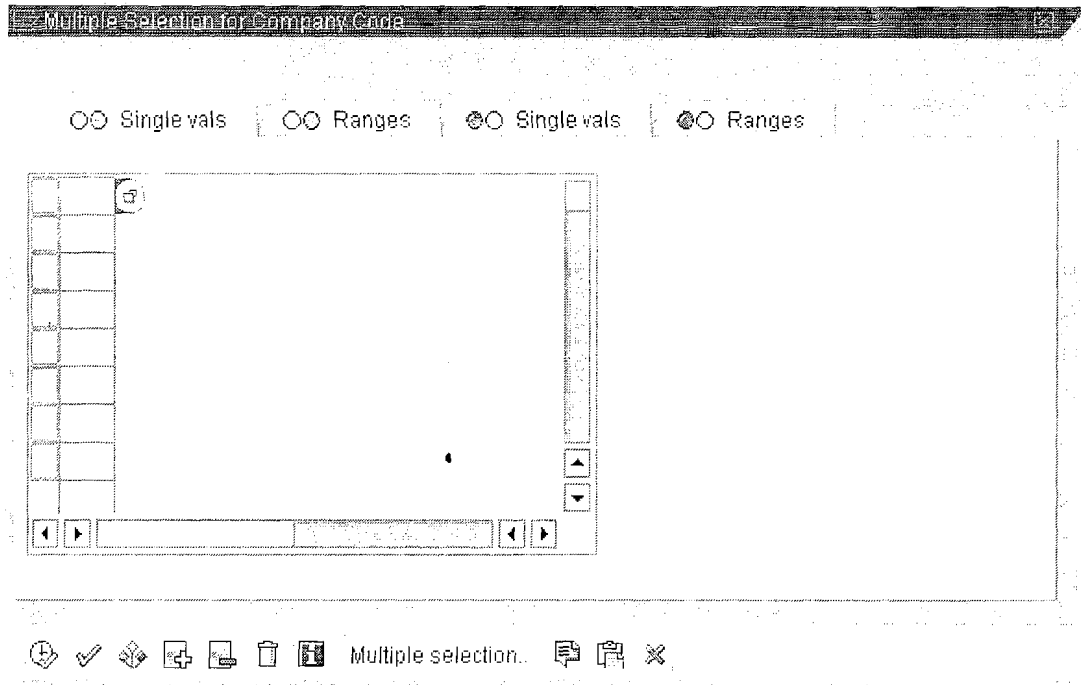
Note: The Value Of the Selected Checkbox/Radiobutton is 'X' and for the Deselected is ' ' (SPACE).

SELECT-OPTIONS to Defining Complex Selections:

Note : PARAMETER Accepts Only Single INPUT and SELECT-OPTIONS to accept Range Of INPUTs on the Selection Screen.



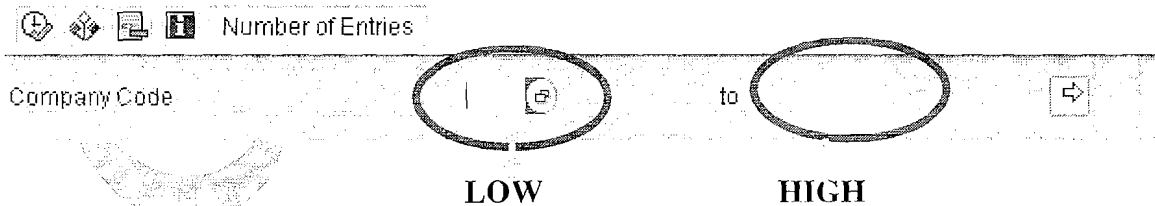
Click On  , to enter Multiple Ranges, Multiple Single Values to be INCLUDED and to be EXCLUDED.



Structure of Selection Table(SELECT-OPTIONS) :

Note : An Internal Table(Selection Table) with header line is Created by the System with the name of the SELECT-OPTIONS and the row type of a selection table is a structure that consists of four components: **LOW, HIGH, SIGN, OPTION.**

Data Browser: Table T001: Selection Screen



SIGN: Possible values are I and E.

- I stands for "inclusive"
- E stands for "exclusive"

OPTION : Possible Operators

- If HIGH is empty, you can use EQ, NE, GT, LE, LT,CP, and NP.
- If HIGH is filled, you can use BT (Between) and NB (Not Between).

SYNTAX :
SELECT-OPTIONS <Name> FOR <Variable>.

Providing Default Values :

```
DATA V_BUKRS TYPE BUKRS.
SELECT-OPTIONS s_bukrs FOR V_BUKRS
                DEFAULT '1000' TO '5000'
                OPTION BT SIGN I.
```

Restricting Entry to One Row

```
SELECT-OPTIONS s_KUNNR for v_KUNNR NO-EXTENSION .
```

Restricting Entry to Single Fields

```
SELECT-OPTIONS s_LIFNR for v_LIFNR for NO INTERVALS .
```

REPORT ZDEMO_SELECT_OPTION .

```
*****
* PROGRAM       :      ZDEMO_SELECT_OPTION
* AUTHOR        :      GANAPATI .ADIMULAM
* START DATE    :      30/06/2006
* PURPOSE       :      IS TO WORK WITH SELECT-OPTIONS
* OPERATIONS
* COPIED FROM   :      NA
*****
* MODIFICATION LOG:
* CHANGE REQUEST :      C11EMAX4756
* -----
*MOD-001       :      DETAILS OF MODIFICATION 001
*MOD-002       :      DETAILS OF MODIFICATION 002
*SUPPLIERS     :      EMAX TECHNOLOGIES
*****
```

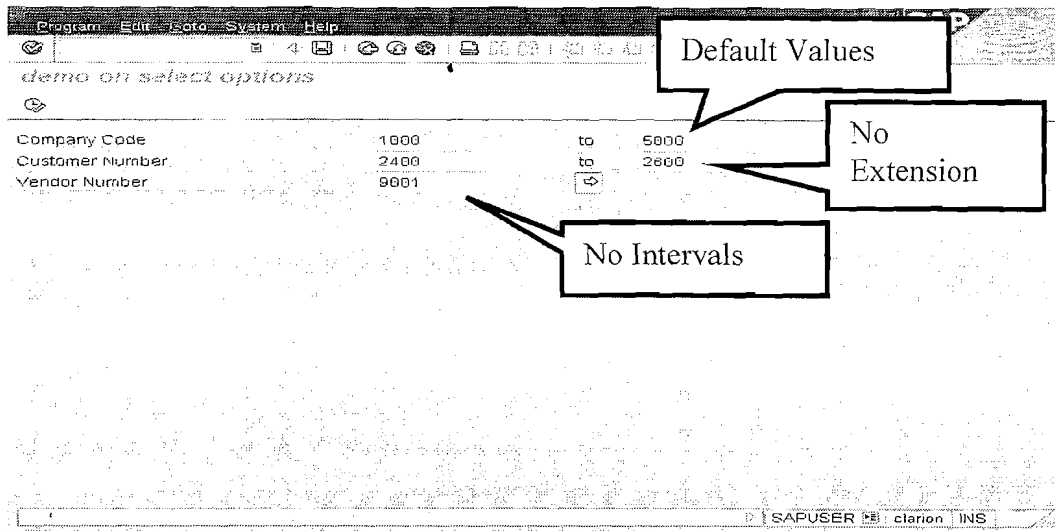
```
DATA : V_BUKRS TYPE BUKRS,
      V_KUNNR TYPE KUNNR,
      V_LIFNR TYPE LIFNR.
```

```
SELECT-OPTIONS S_BUKRS FOR V_BUKRS
                DEFAULT '1000'
                TO '5000'
                OPTION BT
                SIGN I.
```

```
SELECT-OPTIONS S_KUNNR FOR V_KUNNR NO-EXTENSION .
```

```
SELECT-OPTIONS S_LIFNR FOR V_LIFNR NO INTERVALS.
```

OUTPUT OF THE PROGRAM:



Formatting Selection Screens

The selection screen that you define when you use the PARAMETERS or SELECT-OPTIONS statements on their own, has a standard layout in which all parameters **appear line by line**. This layout may not always sufficient.

The SELECTION-SCREEN statement has its own formatting options that you can use to define the layout for selection screens. You can define the layout of parameters and selection criteria and **display comments and underlines** on the selection screen.

Blank Lines:

SELECTION-SCREEN SKIP [<n>].

This statement generates <n> blank lines, where <n> can have a value between 1 and 9. To produce a single blank line, you can omit <n>.

Underlines

To place underlines on the selection screen, you use:

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

This statement generates an underline. If you do not use the <pos(len)> addition, a new line is generated for the underline below the current line, and the underline has the same length as the line. If you use the <pos(len)> addition, the underline begins at **position <pos> in the current line and continues for a length of**

<len> characters. With several elements in one line, you can also specify (<len>) without <pos>. A slash (/) produces a line feed.

Comments :

SELECTION-SCREEN COMMENT [/]<pos(len)> <comm> [FOR FIELD <f>].

SELECTION-SCREEN COMMENT [/]<pos(len)> <comm> [FOR FIELD <f>].

This statement writes the <comm> comment on the selection screen. For <comm>, you can specify a **text symbol**. You must always specify the <pos(len)> addition.

Only if there are several elements in one line, can you omit <pos>.

The text <comm> will be displayed, starting in column <pos>, for a length of <len>. If you do not use a slash (/), the comment is written into the current line; otherwise a new line is created.

You use FOR FIELD <f> to assign a field label to the comment. <f> can be the name of a parameter or a selection criterion(SELECT-OPTION).

Blocks of Elements :

SELECTION-SCREEN BEGIN OF BLOCK <block>
[WITH FRAME [TITLE <title>]].

SELECTION-SCREEN END OF BLOCK <block>.

Note : Make Sure that the title Of the Selection Screen Block should be a TEXT Sysmbol.

Requirement: Design the below Selection Screen AND Process the Resultant Materials according to the Output Selection.

⊕

Material No(s)	
Material NO(s)	to <input type="button" value="↓"/>
Material Types	
Raw	<input checked="" type="radio"/>
Semi Finished	<input type="radio"/>
Finished	<input type="radio"/>
Samples	<input type="radio"/>
Output	
<input type="checkbox"/> Display	
<input type="checkbox"/> File	

PROGRAM

```
*&-----*  
*& Report ZGDEMO_DESIGN_SELECTION_SCREEN *  
*& *  
*&-----*
```

REPORT ZGDEMO_DESIGN_SELECTION_SCREEN

TYPES : BEGIN OF TY_MARA,
 MATNR TYPE MATNR,
 MTART TYPE MTART,
 END OF TY_MARA.

DATA : WA_MARA TYPE TY_MARA,
 IT_MARA TYPE TABLE OF TY_MARA.

DATA V_MATNR TYPE MATNR.
DATA V_MTART TYPE MTART.

SELECTION-SCREEN BEGIN OF BLOCK B1 WITH FRAME TITLE TEXT-001.

SELECT-OPTIONS S_MATNR FOR V_MATNR.

SELECTION-SCREEN END OF BLOCK B1.

Reports

We Never Compromise In Quality, Would You?

SELECTION-SCREEN BEGIN OF BLOCK B2 WITH FRAME TITLE TEXT-002.
PARAMETER : RB_RAW RADIOBUTTON GROUP G1,
 RB_SFİN RADIOBUTTON GROUP G1,
 RB_FIN RADIOBUTTON GROUP G1,
 RB_SAMP RADIOBUTTON GROUP G1.
SELECTION-SCREEN END OF BLOCK B2.

SELECTION-SCREEN BEGIN OF BLOCK B3 WITH FRAME TITLE TEXT-003.
PARAMETER : CB_DISP AS CHECKBOX,
 CB_FILE AS CHECKBOX.
SELECTION-SCREEN END OF BLOCK B3.

```
IF RB_RAW = 'X'. "SELECTED
  V_MTART = 'ROH'.
ELSEIF RB_SFİN = 'X'.
  V_MTART = 'FERT'.
ELSEIF RB_FIN = 'X'.
  V_MTART = 'FERT'.
ELSEIF RB_SAMP = 'X'.
  V_MTART = 'AEM'.
ENDIF.
```

```
SELECT MATNR MTART INTO TABLE IT_MARA
      FROM MARA
      WHERE MATNR IN S_MATNR.
```

```
IF CB_DISP = 'X'.
  WRITE: / 'MATERIAL NOs', 20 'Mat.Type'.
  uline.
  LOOP AT IT_MARA INTO WA_MARA.
    WRITE : / WA_MARA-MATNR, 20 WA_MARA-MTART.
  ENDLOOP.
ELSEIF CB_FILE = 'X'.
```

```
CALL FUNCTION 'GUI_DOWNLOAD'
  EXPORTING
*  BIN_FILESIZE           =
  FILENAME                = 'C:\MATERIAL.TXT'
*  FILETYPE              = 'ASC'
*  APPEND                 = ''
  WRITE_FIELD_SEPARATOR   = 'X'
  TABLES
  DATA_TAB               = IT_MARA
```

Reports
We Never Compromise In Quality, Would You?


```
IF SY-SUBRC = 0.  
  WRITE / 'DATA IS DOWNLOADED SUCCESSFULLY'.  
ENDIF.  
  
ENDIF.
```

OUTPUT OF THE PROGRAM :

EXECUTE THE PROGRAM FOR DISPLAY OPTION :

Demo On Designing the Selection

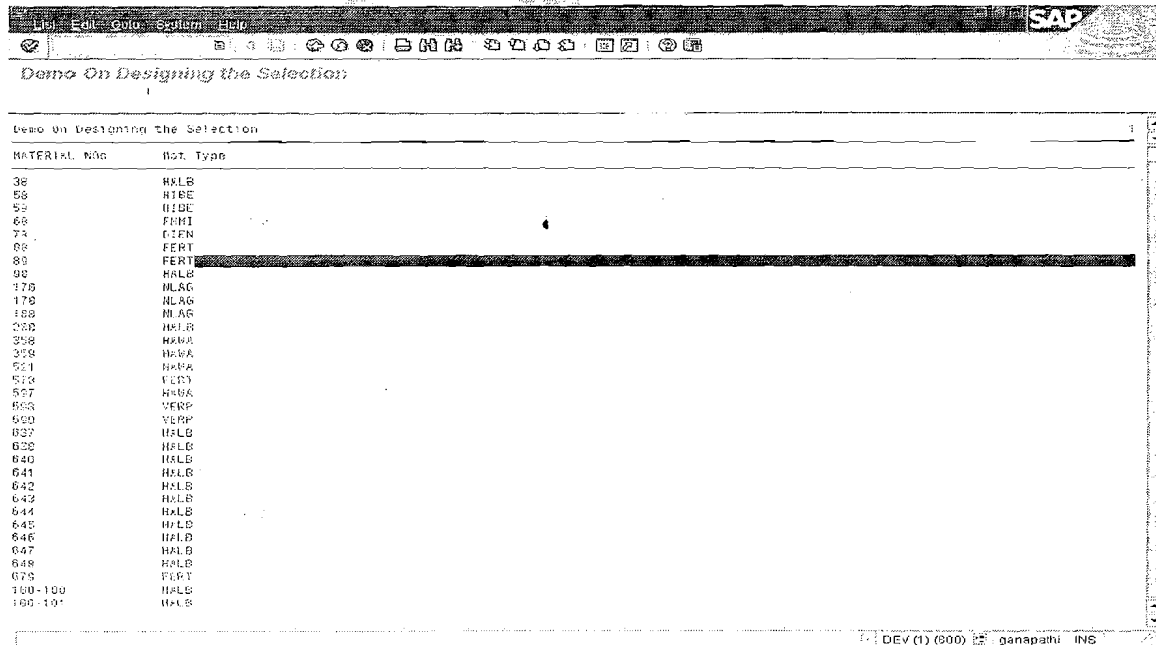
⊕

Material No(s)	
Material NO(s)	to 

Material Types	
Raw	<input type="radio"/>
Semi Finished	<input checked="" type="radio"/>
Finished	<input type="radio"/>
Samples	<input type="radio"/>

Output	
<input checked="" type="checkbox"/> Display	
<input type="checkbox"/> File	


RESULT :




MATERIAL. NO.	Mat. Type
38	HRLB
58	HIBE
59	HIDE
68	FHMI
78	FHFN
80	FERT
80	FERT
98	HALB
178	MLAG
178	MLAG
188	MLAG
288	HSLB
358	HSLB
358	HSLB
521	HSLB
518	FERT
557	HSLB
558	YERP
560	YERP
637	HSLB
638	HSLB
640	HSLB
641	HSLB
642	HSLB
643	HSLB
644	HSLB
645	HSLB
646	HSLB
647	HSLB
648	HSLB
678	FERT
100-100	HSLB
100-101	HSLB

DEV (1) (000) ganapathi INS

EXECUTE THE PROGRAM FOR FILE OPTION



Material No(s)	
Material NO(s)	to 
Material Types	
Raw	<input checked="" type="radio"/>
Semi Finished	<input type="radio"/>
Finished	<input type="radio"/>
Samples	<input type="radio"/>
Output	
<input type="checkbox"/> Display	
<input checked="" type="checkbox"/> File	

OUTPUT

Demo On Designing the Selection

Demo On Designing the Selection

DATA IS DOWNLOADED SUCCESSFULLY

FILE DETAILS(Which is Downloaded):

MATERIAL.TXT - Notepad	
File Edit Format View Help	
0000000000000000000038	HALB
0000000000000000000058	HIBE
0000000000000000000059	HIBE
0000000000000000000068	FHMI
0000000000000000000078	DIEN
0000000000000000000088	FERT
0000000000000000000089	FERT
0000000000000000000098	HALB
0000000000000000000170	NLAG
0000000000000000000178	NLAG
0000000000000000000188	NLAG
0000000000000000000288	HALB
0000000000000000000358	HAWA
0000000000000000000359	HAWA
0000000000000000000521	HAWA

CLASSICAL REPORT EVENTS :-

DAY-2

Name of the Event	Triggering	Purpose
INITIALIZATION	This event occurs before the standard selection screen is called.	To initialize the input fields of the standard selection screen
AT SELECTION-SCREEN	This event is processed before leaving the Selection Screen i.e. when the selection screen has been processed (at the end of PAI once the ABAP runtime environment has passed all of the input data from the selection screen to the ABAP program).	To Validate the input provided through Selection Screen. Note: If an error message occurs in this processing block, the selection screen is redisplayed with all of its fields ready for input. This allows you to check input values for consistency.
AT SELECTION-SCREEN ON <Field Name>.	This event is processed before leaving the Selection Screen Element.	To Validate the Individual input provided through Selection Screen. Note: It re displays only the this Particular Field if the input is wrongly entered.
START-OF-SELECTION	This event occurs after the selection screen has been processed and before data is read using the logical database.	Note:- The REPORT statement always executes a START-OF-SELECTION implicitly consequently all processing logic i.e, non-declarative statements that occur between the REPORT or PROGRAM statement and the first processing block are also processed in the START-OF-SELECTION block.
TOP-OF-PAGE	This is a list processing event executed before the first data is output on	This allows you to define output which supplements the standard page header at

Reports
We Never Compromise In Quality, Would You?

	<p>a new page. This is processed only when generating basic list. This is only executed before outputting the first line using any output statement such as write , uline, skip on a new page. Note: It is not triggered by a NEW-PAGE statement.</p>	<p>the beginning of the page.</p>
END-OF-PAGE	<p>This will be triggered when it reaches the End-Of-Page.</p>	<p>This is used to print the same Footer details for all the pages.</p>
END-OF-SELECTION	<p>This is the last of the events called by the runtime environment to occur. It is triggered after all of the data has been read from the logical database, and before the list processor is started.</p>	<p>Used to print the final output . Used to print Grand Totals when working with Logical Database.</p>

Example for initialization:

To change a selection criterion, you must fill at least the components <seltab>-SIGN, <seltab>-OPTION, and <seltab>-LOW of the selection table <seltab>, otherwise it remains undefined.

Note : Initialization event is processed only once.

EXAMPLE PROGRAM ON CLASSICAL REPORT EVENTS

```
*****
* PROGRAM       : ZDEMO_CLASSICAL_REPORT_EVENTS
* AUTHOR        : GANAPATI .ADIMULAM
* START DATE    : 27/01/2008
* PURPOSE       : REPORT TO DISPLAY THE PURCHASE DOCS
                  USING EVENTS
* COPIED FROM   : NA
*****
* MODIFICATION LOG:
* CHANGE REQUEST : C11EMAX4756
*-----
*MOD-001       : DETAILS OF MODIFICATION 001
*MOD-002       : DETAILS OF MODIFICATION 002
*SUPPLIERS     : EMAX TECHNOLOGIES
*****
```

REPORT ZDEMO_CLASSICAL_REPORT_EVENTS NO STANDARD PAGE
HEADING LINE-COUNT 20(3).

TYPES : BEGIN OF TY_POS,
 EBELN TYPE EBELN,
 LIFNR TYPE ELIFN,
 BUKRS TYPE BUKRS,
 EKORG TYPE EKORG,
 EBELP TYPE EBELP,
 MATNR TYPE MATNR,
 MENGE TYPE BSTMG,
 NETPR TYPE BPREI,
END OF TY_POS.

DATA : WA_POS TYPE TY_POS,
 IT_POS TYPE TABLE OF TY_POS.
DATA : V_BUKRS TYPE BUKRS,
 V_EKORG TYPE EKORG.

SELECTION-SCREEN BEGIN OF BLOCK B1 WITH FRAME.
SELECT-OPTIONS: S_BUKRS FOR V_BUKRS, "COMPANY CODE
 S_EKORG FOR V_EKORG. "PURCHASING ORGANIZATION
SELECTION-SCREEN END OF BLOCK B1.

* TOP-OF-PAGE

TOP-OF-PAGE.
WRITE : /5 'Comp.Code',
 17 'Vendor',
 28 'Pur.Org',
 40 'Doc.No',
 52 'Item',
 60 'Mat.No',
 80 'Price'.

ULINE.

* INITIALIZATION

INITIALIZATION.
S_BUKRS-LOW = '1000'.
S_BUKRS-HIGH = '5000'.
S_BUKRS-SIGN = 'T'.
S_BUKRS-OPTION = 'BT'.
APPEND S_BUKRS.

* AT SELECTION-SCREEN

AT SELECTION-SCREEN ON S_BUKRS.
SELECT BUKRS INTO V_BUKRS FROM T001 WHERE BUKRS IN
S_BUKRS.
ENDSELECT.
IF SY-SUBRC <> 0.
WRITE : / 'INVALID COMPANY CODE'.
ENDIF.
AT SELECTION-SCREEN ON S_EKORG.
SELECT EKORG INTO V_EKORG FROM EKKO WHERE EKORG IN
S_EKORG.
ENDSELECT.
IF SY-SUBRC <> 0.
WRITE : / 'INVALID PURCHASING ORGANIZATION'.
ENDIF.

* START-OF-SELECTION.

START-OF-SELECTION.

SELECT EKKO~EBELN

EKKO~LIFNR

EKKO~BUKRS

EKKO~EKORG

EKPO~EBELP

EKPO~MATNR

EKPO~MENGE

EKPO~NETPR

INTO TABLE IT_POS

FROM EKKO INNER JOIN EKPO

ON EKKO~EBELN = EKPO~EBELN

WHERE EKKO~BUKRS IN S_BUKRS

AND EKKO~EKORG IN S_EKORG.

* END-OF-SELECTION.

END-OF-SELECTION.

IF NOT IT_POS IS INITIL.

LOOP AT IT_POS INTO WA_POS.

WRITE : /5 WA_POS-BUKRS,

17 WA_POS-LIFNR,

28 WA_POS-EKORG,

40 WA_POS-EBELN,

52 WA_POS-EBELP,

60 WA_POS-MATNR,

80 WA_POS-NETPR.

CLEAR WA_POS.

ENDLOOP.

ELSE.

WRITE : /' NO RECORDS FOUND FOR THE GIVEN SELECTION
CRITERIA'.

ENDIF.

* END-OF-PAGE

END-OF-PAGE.

ULINE.

WRITE : /30 'eMAX Technologies,Ameerpet,Ph-66972767'.

ULINE.

Reports
We Never Compromise In Quality, Would You?

OUTPUT:

Program Edit Goto System Help

Report to display the Purchasing Documents

Company Code 1000 to 5000
Purchasing Organisation 1000 to 5000

List Edit Goto System Help

Report to display the Purchasing Documents

Comp. Code	Vendor	Pur. Org	Doc. No	Item	Mat. No	Price
1000	1101	1000	6000000004	00010		11,600,00
1000	1102	1000	6000000005	00010		10,300,00
1000	1103	1000	6000000006	00010		10,700,00
1000	1000	1000	414-0100	00010	500-120	2,47
1000	1000	1000	414-0100	00020	500-130	3,95
1000	1000	1000	414-0100	00030	500-140	1,21
1000	1000	1000	414-0100	00040	500-150	2,04
1000	1000	1000	414-0100	00050	500-160	1,28
1000	1000	1000	414-0100	00060	500-170	1,51
1000	1000	1000	414-0100	00070	500-180	1,78
1000	1000	1000	414-0100	00080	500-190	1,67
1000	1000	1000	414-0100	00090	500-200	1,01
1000	1000	1000	414-0100	00100	500-210	0,93
1000	1000	1000	414-0100	00110	500-220	0,95
1000	1000	1000	414-0101	00010	500-120	2,57

eMAX Technologies,Amcerpet,Ph-66972767

Control level Processing(Control Break Statements) :

Control level processing is allowed within a LOOP over an internal table. This means that you can divide sequences of entries into groups based on the contents of certain fields.

Internal tables are divided into groups according to the sequence of the fields in the line structure. The first column defines the highest control level and so on. **The control level hierarchy must be known when you create the internal table.**

The control levels are formed by sorting the internal table in the sequence of its structure, that is, by the first field first, then by the second field, and so on. Tables in which the table key occurs at the start of the table are particularly suitable for control level processing.

The following diagram illustrates control level processing in a sorted table, where different field contents in the first three fields are indicated by different colors:

The AT statement introduces a statement block that you end with the ENDAT statement.

AT <Level>.

<Statement Block>.

ENDAT.

You can react to the following control level changes:

<level>	Meaning
AT FIRST	First line of the internal table
AT LAST	Last line of the internal table
AT NEW <f>	Beginning of a group of lines with the same contents in the field <f> and in the fields left of <f>
AT END Of <f>	End of a group of lines with the same contents in the field <f> and in the fields left of <f>

You can use control level statements to react to control breaks in internal tables instead of programming them yourself with logical expressions. Within the loop, you must order the AT-ENDAT statement blocks according to the hierarchy of the control levels. If the internal table has the columns <f₁>, <f₂>,, and if it is sorted by these columns, you must program the loop as follows:

LOOP AT <itab>.
AT FIRST. ... ENDAT.
AT NEW <f1>. ENDAT.
T NEW <f₂>. ENDAT.

.....
<single line processing>

.....
AT END OF <f2>. ... ENDAT.
AT END OF <f1>. ... ENDAT.
AT LAST. ENDAT.
ENDLOOP.

The innermost hierarchy level <single line processing> processes the table lines that do not correspond to a control level change. You do not have to use all control level statements. But you must place the used ones in the above sequence. You should not use control level statements in loops where the line selection is restricted by WHERE or FROM and TO. Neither should the table be modified during the loop.

If you are working with a work area <wa>, it **does not** contain the current line in the AT... ENDAT statement block. **All character fields to the right of the current group key are filled with asterisks (*). All other fields to the right of the current group key contain their initial value.**

Within an AT...ENDAT block, you can calculate the contents of the numeric fields of the corresponding control level using the SUM statement.

SUM.

You can only use this statement within a LOOP. If you use SUM in an AT - ENDAT block, the system calculates totals for the numeric fields of **all** lines in the current line group and writes them to the corresponding fields in the work area (see example in). If you use the SUM statement outside an AT - ENDAT block (single entry processing), the system calculates totals for the numeric fields of **all** lines of the internal table in **each** loop pass and writes them to the corresponding fields of the work area. It therefore only makes sense to use the SUM statement in AT...ENDAT blocks.

If the table contains a nested table, you cannot use the SUM statement. Neither can you use it if you are using a field symbol instead of a work area in the LOOP statement.

EXAMPLE PROGRAM

REPORT ZDEMO_CONTROL_BREAK_EVENTS line-size 200.

* PROGRAM : ZDEMO_CONTROL_BREAK_EVENTS
* AUTHOR : GANAPATI.ADIMULAM
* START DATE : 27/01/2008
* PURPOSE : CTRL BREAK STATEMENTS
* OPERATIONS
* COPIED FROM : NA

* MODIFICATION LOG:

* CHANGE REQUEST : C11EMAX4756
* MOD-001 : DETAILS OF MODIFICATION 001
* MOD-002 : DETAILS OF MODIFICATION 002
* SUPPLIERS : EMAX TECHNOLOGIES

DATA : V_EKORG TYPE EKKO-EKORG,
V_BUKRS TYPE BUKRS.

SELECTION-SCREEN BEGIN OF BLOCK B1 WITH FRAME TITLE TEXT-000.

SELECT-OPTIONS : S_BUKRS FOR V_BUKRS,
S_EKORG FOR V_EKORG.
SELECTION-SCREEN END OF BLOCK B1.

TYPES : BEGIN OF TY_EKKO,
BUKRS TYPE BUKRS,
EKORG TYPE EKORG,
EBELN TYPE EBELN,
LIFNR TYPE ELIFN,
ANGNR TYPE ANGNR,
VERKF TYPE EVERK,
LLIEF TYPE LLIEF,
INCO1 TYPE INCO1,
INCO2 TYPE INCO2,
EBELP TYPE EBELP,
MATNR TYPE MATNR,
WERKS TYPE EWERK,
LGORT TYPE LGORT_D,
MATKL TYPE MATKL,
NETWR TYPE BWERT,
END OF TY_EKKO.

Reports
We Never Compromise In Quality, Would You?

DATA : IT_EKKO TYPE STANDARD TABLE OF TY_EKKO,
WA_EKKO TYPE TY_EKKO.

SELECT

EKKO~BUKRS
EKKO~EKORG
EKKO~EBELN
EKKO~LIFNR
EKKO~ANGNR
EKKO~VERKF
EKKO~LLIEF
EKKO~INCO1
EKKO~INCO2
EKPO~EBELP
EKPO~MATNR
EKPO~WERKS
EKPO~LGORT
EKPO~MATKL
EKPO~NETWR

INTO TABLE IT_EKKO
FROM EKKO INNER JOIN EKPO
ON EKKO~EBELN = EKPO~EBELN
WHERE EKKO~BUKRS IN S_BUKRS AND
EKKO~EKORG IN S_EKORG AND
EKKO~BSTYP = 'F'.

IF SY-SUBRC = 0.
WRITE : /5 'BUKRS', 16 'EKORG', 24 'PO', 35 'ITEM',50 'AMOUNT'.
ULINE.

SORT IT_EKKO BY BUKRS EKORG EBELN.

LOOP AT IT_EKKO INTO WA_EKKO.
WRITE : / WA_EKKO-BUKRS UNDER 'BUKRS',
WA_EKKO-EKORG UNDER 'EKORG',
WA_EKKO-EBELN UNDER 'PO',
WA_EKKO-EBELP UNDER 'ITEM',
WA_EKKO-NETWR UNDER 'AMOUNT'.

AT END OF EBELN.
SUM.
ULINE.
WRITE : /40 'SUM OF EACH PO', WA_EKKO-EBELN, WA_EKKO-NETWR.
ULINE.
ENDAT.

Reports
We Never Compromise In Quality, Would You?

AT END OF EKORG.

SUM.

SKIP.

WRITE : /40 'SUM OF EACH EKORG',WA_EKKO-BUKRS,
WA_EKKO-EKORG,WA_EKKO-NETWR.

SKIP.

ENDAT.

AT END OF BUKRS.

SUM.

SKIP.

WRITE : /40 'SUM OF EACH BUKRS',WA_EKKO-BUKRS,WA_EKKO-
NETWR.

SKIP.

ENDAT.

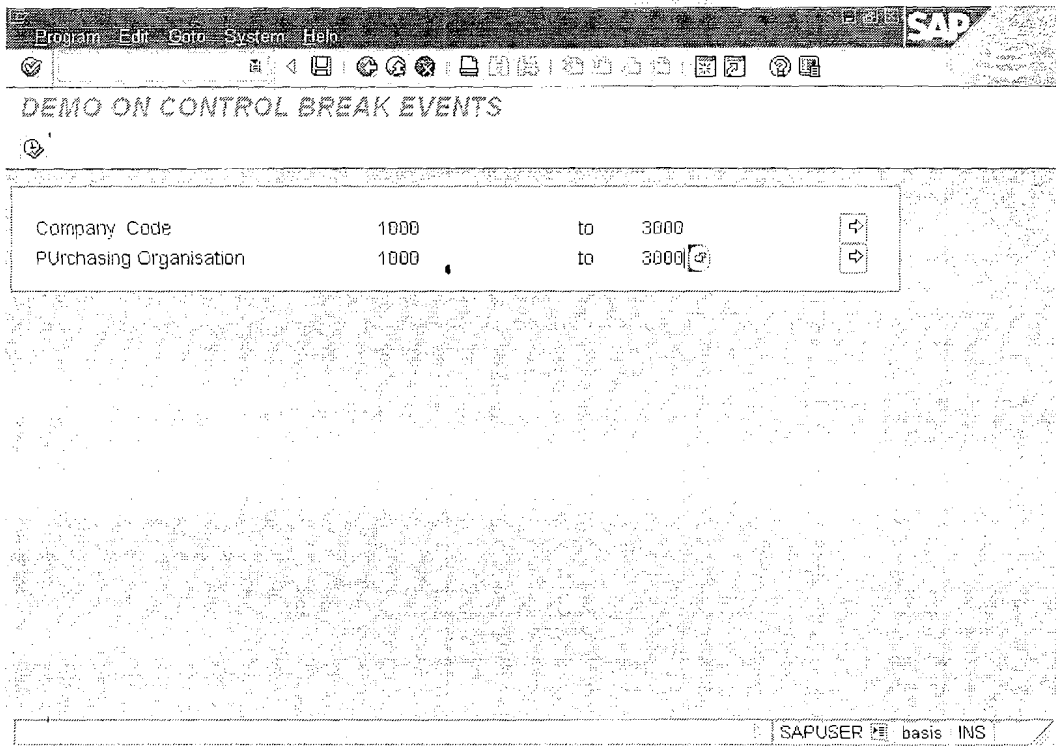
CLEAR : WA_EKKO.

ENDLOOP.

ELSE.

WRITE : / 'NO RECORDS FOUND FOR THE GIVEN SELECTION'.

ENDIF.



Reports
We Never Compromise In Quality, Would You?

SAP
 List Edit Cont System Help
 [Icons]

DEMO ON CONTROL BREAK EVENTS

BUKRS	EKORG	PO	Item	AMOUNT
1000	1000	414-0100	00070	1.602,00
1000	1000	414-0100	00080	1.338,00
1000	1000	414-0100	00090	808,00
1000	1000	414-0100	00100	837,00
1000	1000	414-0100	00110	760,00
1000	1000	414-0100	00060	1.812,00
1000	1000	414-0100	00010	3.705,00
1000	1000	414-0100	00020	4.575,00
1000	1000	414-0100	00030	1.684,00
1000	1000	414-0100	00040	2.856,00
1000	1000	414-0100	00050	1.536,00
Sum Of Each PO 414-0100				21.521,00
1000	1000	414-0101	00070	1.710,00
1000	1000	414-0101	00080	1.408,00
1000	1000	414-0101	00090	872,00
1000	1000	414-0101	00100	909,00

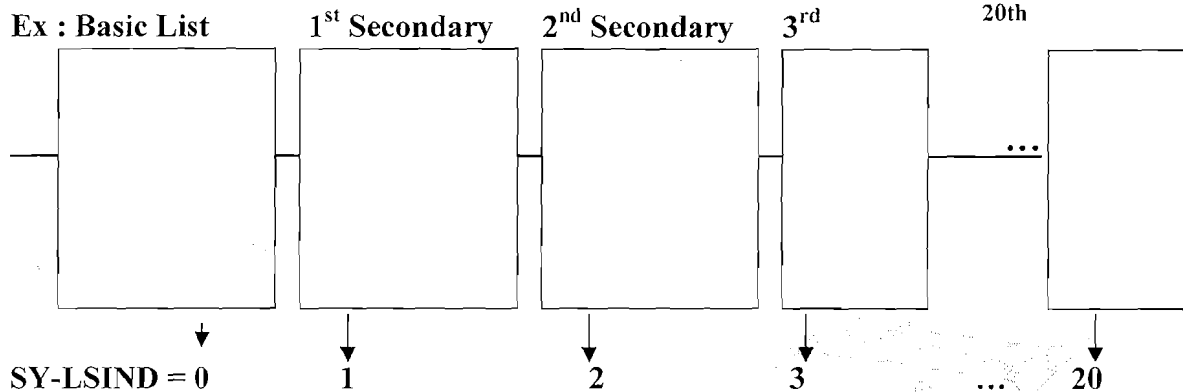
SAPUSER basis INS



Interactive Reports

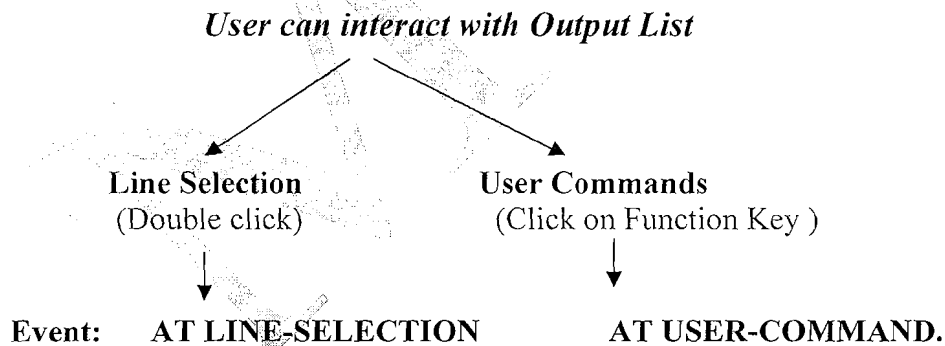
DAY-4

Display the Summarized Information as the First List_ And letting the USER Interact for the Detailed Info.



NOTE: We Can go up to 20 Secondary Lists.

Different Ways Of User Interaction and the Corresponding Events Triggered for Each type Of User Interaction



Note: The List Index SY-LSIND will be incremented by 1 for each user Interaction.

It is ZERO for Basic List , 1 for 1st 2nd ry, 2 for 2nd ry etc. upto 20
As We Can Generate Upto 20 Levels.

By default, the basic list has a standard list status and a standard page header. The TOP-OF-PAGE and END-OF-PAGE events can occur while the **basic list** is being created.

Working With AT LINE-SELECTION

It Triggers When the User Interacts with the Output Line (Line Selection/Double Click).

Useful System Fields for Details Lists

After each user action on a list, the following ABAP system fields will be set in the corresponding event block:

System field	Information
SY-LSIND	Index of the list created during the current event (basic list = 0)
SY-LISTI	Index of the list level from which the event was triggered
SY-LILLI	Absolute number of the line from which the event was triggered
SY-LISEL	Contents of the line from which the event was triggered
SY-CUROW	Position of the line in the window from which the event was triggered (counting starts with 1)
SY-CUCOL	Position of the column in the window from which the event was triggered (counting starts with 2)
SY-CPAGE	Page number of the first displayed page of the list from which the event was triggered
SY-STARO	Number of the first line of the first page displayed of the list from which the event was triggered (counting starts with 1). This line may contain the page header.
SY-STACO	Number of the first column displayed in the list from which the event was triggered (counting starts with 1)

Reports
We Never Compromise In Quality, Would You?

SY-UCOMM	Function code that triggered the event
SY-PFKEY	Status of the list currently being displayed.

Note : Each 2nd ry List is a normal output list Only. But the Next Level Data depends on the Selected Line and Contents from the Previous List. So it is enough to know the way to find out the Selected Line Details of the Previous List.

Ways to Find out the Selected Line Details from the Previous List (OR)

Passing Data by Program Statements :

SY-LISEL : Note : SY-LISEL is the System Variable to Maintain, the Entire Selected Line Contents and we need to split it for the required Field Contents.

HIDE : The HIDE statement is one of the fundamental statements for interactive reporting. You use the HIDE technique when creating a basic list. It defines the information that can be passed to subsequent detail lists.

GET CURSOR : Use the statements GET CURSOR FIELD and GET CURSOR LINE to pass the output field or output line on which the cursor was positioned during the interactive event to the ABAP program.

WORKING WITH SY-LISEL :

Note : SY-LISEL is the System Variable to Maintain, the Entire Selected Line Contents and we need to split it for the required Field Contents.

EXAMPLE PROGRAM :

```
REPORT ZGDEMO_INTERACTIIVE_SY_LISEL
*****
*                               Program Heading
*****
*AUTHOR       : GANAPATI.ADIMULAM
*PROGRAM      : ZGDEMO_INTERACTIIVE_SY_LISEL
*COPIED FROM  : NA
*PURPOSE      : DISPLAY THE COMPANY DETAILS AS BASIC LIST
*              AND CUSTOMER DETAILS AND Reconciliation
*              Account As Secondary Details
*REFERENCE DOCS : DEVWR0065
*****
*MODIFICATION LOG :
*MOD-0001        : DETAILS ABOUT THE FIRST CHANGE IN THE PROG
*MOD-002         : DETAILS ABOUT THE 2ND CHANGE IN THE PROG
*TRANSPORT REQ.NO : C11DEV12354
*****
```

Reports
We Never Compromise In Quality, Would You?

```
TYPES : BEGIN OF TY_T001,
        BUKRS TYPE BUKRS, "COMPANY CODE
        BUTXT TYPE BUTXT, "NAME OF THE COMPANY
        ORT01 TYPE ORT01, "CITY
        LAND1 TYPE LAND1, "COUNTRY KEY
        WAERS TYPE WAERS, "CURRENCY KEY
        END OF TY_T001.
DATA : WA_T001 TYPE TY_T001,
        IT_T001 TYPE STANDARD TABLE OF TY_T001.
```

```
DATA : V_BUKRS TYPE BUKRS.
```

```
*CUSTOMER MASTER DATA
TYPES : BEGIN OF TY_CUSTOMER,
        KUNNR TYPE KUNNR, "CUSTOMER NO
        BUKRS TYPE BUKRS, "COMPANY CODE
        AKONT TYPE AKONT, "RECONCILIATION ACCOUNT
        END OF TY_CUSTOMER.
```

```
DATA : IT_CUSTOMER TYPE STANDARD TABLE OF TY_CUSTOMER,
        WA_CUSTOMER TYPE TY_CUSTOMER.
```

```
*DESIGNING THE SELECTION SCREEN
SELECT-OPTIONS : S_BUKRS FOR V_BUKRS.
*****
```

```
*          START-OF-SELECTION.          *
*****
```

```
START-OF-SELECTION.
SELECT  BUKRS "COMPANY CODE
        BUTXT "NAME OF THE COMPANY
        ORT01 "CITY
        LAND1 "COUNTRY KEY
        WAERS "CURRENCY KEY
        INTO TABLE IT_T001
        FROM T001
        WHERE BUKRS IN S_BUKRS.
```

```
*****
*          END-OF-SELECTION.          *
*****
```

```
END-OF-SELECTION.

IF NOT IT_T001 IS INITIAL.
  WRITE : /5 'COM.CODE', 15 'TEXT', 45 'CITY', 70 'COUNTRY'.
  ULINE.
```

Reports

We Never Compromise In Quality, Would You?

LOOP AT IT_T001 INTO WA_T001.

```
WRITE :/5 WA_T001-BUKRS,  
      15 WA_T001-BUTXT,  
      45 WA_T001-ORT01,  
      70 WA_T001-LAND1.
```

```
CLEAR WA_T001.  
ENDLOOP.
```

```
ENDIF.
```

```
*****
```

```
* AT LINE-SELECTION.
```

```
*****
```

```
AT LINE-SELECTION.
```

```
CASE SY-LSIND.
```

```
  WHEN 1.
```

```
    V_BUKRS = SY-LISEL+4(4).
```

```
*RETRIEVE ALL THE CUSTOMERS FOR THE SELECTED COMPANY CODE
```

```
  SELECT KUNNR "CUSTOMER  
         BUKRS "COMPANY CODE  
         AKONT "RECON.ACC  
  INTO TABLE IT_CUSTOMER  
  FROM KNB1 WHERE BUKRS = V_BUKRS.
```

```
*DISPLAY THE DATA
```

```
  IF NOT IT_CUSTOMER IS INITIAL.
```

```
    WRITE :/5 'COM.CODE', 15 'CUSTOMER',30 'RECON ACC'.
```

```
    ULINE.
```

```
    LOOP AT IT_CUSTOMER INTO WA_CUSTOMER.
```

```
      WRITE :/5 WA_CUSTOMER-BUKRS, "COMPANY CODE
```

```
              15 WA_CUSTOMER-KUNNR, "CUSTOMER
```

```
              30 WA_CUSTOMER-AKONT. "RECONCILIATION ACCOUNT
```

```
      CLEAR WA_CUSTOMER.
```

```
    ENDLOOP.
```

```
  ELSE.
```

```
    WRITE :/ 'NO FURTHER DETAILS FOUND FOR THE SELECTED  
COMP.CODE'.
```

```
  ENDIF.
```

```
ENDCASE.
```

SY-LISEL+4(4) gives the Company Code Of the Selected Line. Because BUKRS was Printed as **WRITE: /5 WA_T001-BUKRS.** Since String starts at ZERO Place , +4(4) = 4 Chars from 5th Position.

EXECUTE THE PROGRAM :

Demo to Work With SY-LISEL



S_BUKRS _____ to _____

EXECUTE

COM CODE	TEXT	CITY	COUNTRY
0001	SAP A.G	Walldorf	DE
1000	IDES AD	Frankfurt	DE
2000	IDES UK	London	GB
2100	IDES Portugal	Lisbon	PT
2200	IDES France	Paris	FR
2500	IDES España	Barcelona	ES
2400	IDES Italia	Milano	IT
2500	IDES Netherlands	Rotterdam	NL
3000	IDES US INC	New York	US
3010	Durg Subsidiary - Belgium	Brussels	BE
4000	IDES Canada	Toronto	CA
4500	Canadian Company	Toronto	CA
5000	IDES Japan	Tokyo	JP
5800	IDES Mexico, S.A. de C.V.	México DF	MX
7000	IDES Brazil	São Paulo	BR
7500	IDES Argentina	Buenos Aires	AR
7800	IDES Columbia	Columbia	CO
7700	IDES Venezuela	Venezuela	VE
7800	IDES Peru	Perú	PE
8000	IDES Chile	Chile	CL
AT01	IDES AB & Co. KG	Frankfurt	DE
CA00	Walling	SAPLand	DE
CA00	CA00	SAPLand	DE
CH01	FIRB Schweiz	Schweiz	CH
CP00	Good Food	Chicago	US
F100	Bankhaus Frankfurt	Frankfurt	DE
F300	Liberty Bank	New York	US
R100	IDES Retail B&H	Boston	DE
R300	IDES Retail INC US	Los Angeles	US
S300	IDES Services	Atlanta	US

Double Click On Company Code 0001, and Notice that all the Customers from Company Code 0001 are Displayed.

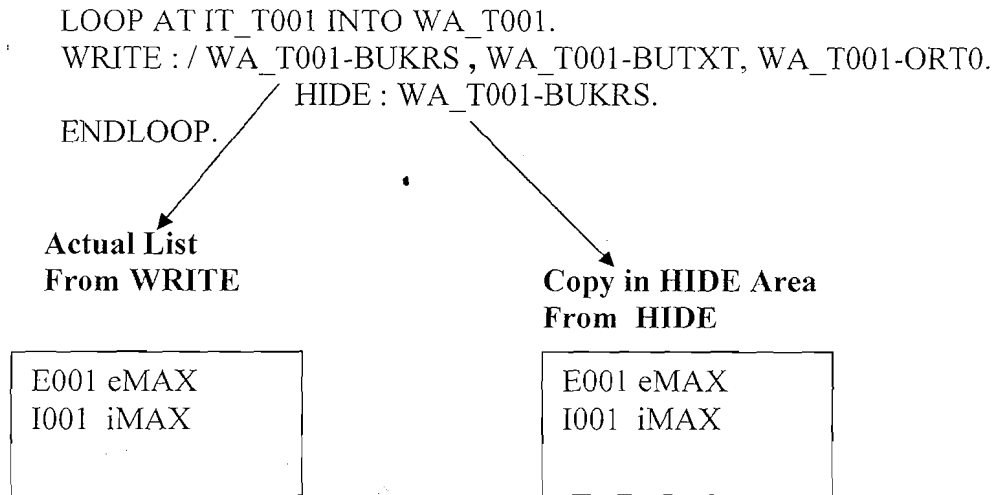
Demo to Work With SY-LISEL



COM CODE	CUSTOMER	RECON ACC
0001	1000	140000
0001	1050	140000
0001	2000	141000
0001	2222	141000
0001	1000000015	140000

WORKING WITH HIDE TECHNIQUE:

You use the HIDE technique while creating a list level to store line-specific information for later use.



NOTE: HIDE should be always after the output (write) statement.. Hide maintains the copy of the output(corresponding) list.

NOTE:When the user interacts with any line from any level,the system checks for the corresponding 'HIDE' area and Picks up the corresponding selected line contents, If HIDE Area Found, it stores the selected line Contents Back from HIDE Area to the variables(WA) through which the **HIDE** area is generated.

EXAMPLE PROGRAM Using HIDE :

```

REPORT ZDEMO_INTERACTIVE_HIDE.
*****
*           Program Heading
*****
*AUTHOR      : GANAPATI.ADIMULAM
*PROGRAM     : ZDEMO_INTERACTIVE_HIDE
*COPIED FROM : NA
*PURPOSE     : DISPLAY THE COMPANY DETAILS ON BASIC LIST
*             AND CUSTOMER DETAILS AND Reconciliation
*             Account: IS DEVELOPED USING HIDE
*
*REFERENCE DOCS : DEVWR0065
*****
*MODIFICATION LOG :
*MOD-0001      : DETAILS ABOUT THE FIRST CHANGE IN THE PROG
*MOD-002      : DETAILS ABOUT THE 2ND CHANGE IN THE PROG
    
```

Reports
We Never Compromise In Quality, Would You?

*TRANSPORT REQ.NO : C11DEV12354

TYPES : BEGIN OF TY_T001,
 BUKRS TYPE BUKRS, "COMPANY CODE
 BUTXT TYPE BUTXT, "NAME OF THE COMPANY
 ORT01 TYPE ORT01, "CITY
 LAND1 TYPE LAND1, "COUNTRY KEY
 WAERS TYPE WAERS, "CURRENCY KEY
END OF TY_T001.

DATA : IT_T001 TYPE STANDARD TABLE OF TY_T001,
 WA_T001 TYPE TY_T001.

DATA : V_BUKRS TYPE BUKRS.

*CUSTOMER MASTER DATA
TYPES : BEGIN OF TY_CUSTOMER,
 KUNNR TYPE KUNNR, "CUSTOMER NO
 BUKRS TYPE BUKRS, "COMPANY CODE
 AKONT TYPE AKONT, "RECONCILIATION ACCOUNT
END OF TY_CUSTOMER.

DATA : IT_CUSTOMER TYPE STANDARD TABLE OF TY_CUSTOMER,
 WA_CUSTOMER TYPE TY_CUSTOMER.

*DESIGNING THE SELECTION SCREEN
SELECT-OPTIONS : S_BUKRS FOR V_BUKRS.

* START-OF-SELECTION. *

START-OF-SELECTION.
SELECT BUKRS "COMPANY CODE
 BUTXT "NAME OF THE COMPANY
 ORT01 "CITY
 LAND1 "COUNTRY KEY
 WAERS "CURRENCY KEY
INTO TABLE IT_T001
FROM T001
WHERE BUKRS IN S_BUKRS.

* END-OF-SELECTION. *

END-OF-SELECTION.

IF NOT IT_T001 IS INITIAL.

WRITE : /5 'COM.CODE', 15 'TEXT', 45 'CITY', 70 'COUNTRY'.
ULINE.

LOOP AT IT_T001 INTO WA_T001.

WRITE : /5 WA_T001-BUKRS,
15 WA_T001-BUTXT,
45 WA_T001-ORT01,
70 WA_T001-LAND1.

HIDE : WA_T001-BUKRS,
WA_T001-BUTXT,
WA_T001-ORT01,
WA_T001-LAND1.

CLEAR WA_T001.
ENDLOOP.

ENDIF.

* AT LINE-SELECTION. *

AT LINE-SELECTION.

CASE SY-LSIND.

WHEN 1.

*RETRIEVE ALL THE CUSTOMERS FOR THE SELECTED COMPANY
CODE

SELECT KUNNR "CUSTOMER
BUKRS "COMPANY CODE
AKONT "RECON.ACC
INTO TABLE IT_CUSTOMER
FROM KNB1 WHERE BUKRS = WA_T001-BUKRS.

*DISPLAY THE DATA

IF NOT IT_CUSTOMER IS INITIAL.

WRITE : /5 'COM.CODE', 15 'CUSTOMER', 30 'RECON ACC'.

ULINE.

LOOP AT IT_CUSTOMER INTO WA_CUSTOMER.

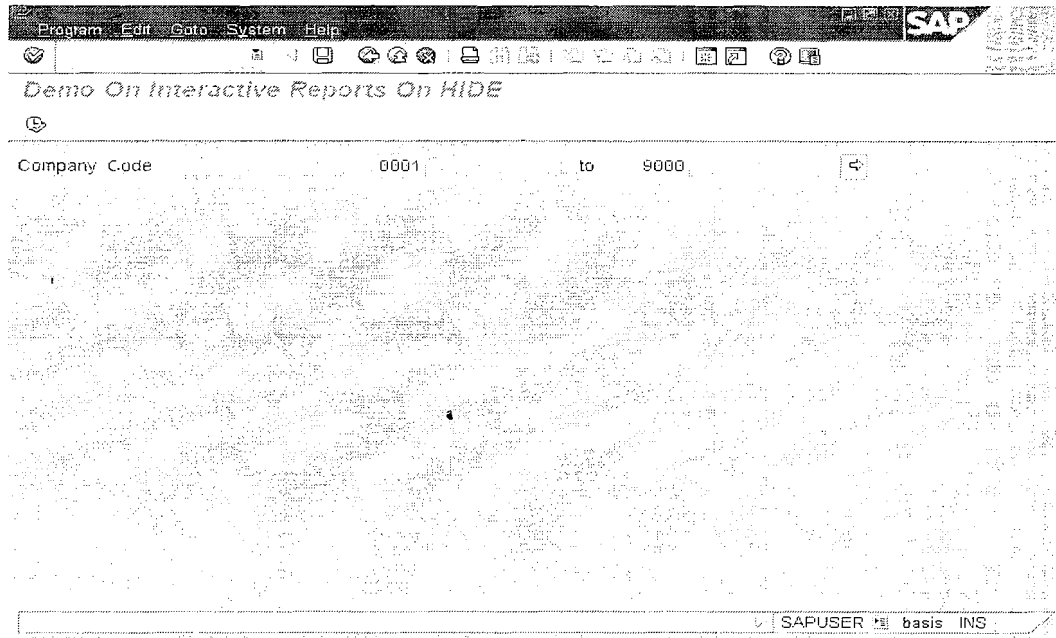
WRITE : /5 WA_CUSTOMER-BUKRS, "COMPANY CODE
15 WA_CUSTOMER-KUNNR, "CUSTOMER
30 WA_CUSTOMER-AKONT. "RECONCILIATION ACCOUNT

CLEAR WA_CUSTOMER.
ENDLOOP.

Reports
We Never Compromise In Quality, Would You?

```
ELSE.  
WRITE :/ 'NO FURTHER DETAILS FOUND FOR THE SELECTED  
COMP.CODE'.  
  
ENDIF.  
  
ENDCASE.
```

EXECUTE THE PROGRAM:



OUTPUT:

List Edit Goto System Help

Demo On Interactive Reports On HIDE

Demo On Interactive Reports On HIDE

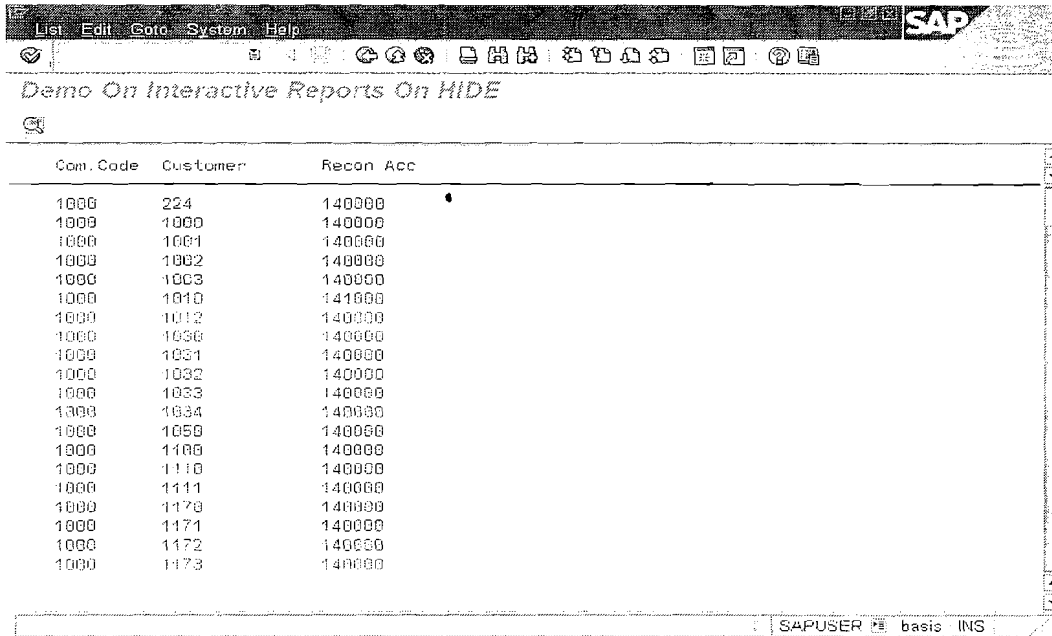
Com. Code	Text	City	Country
0001	SAP A.G.	Walldorf	DE
1000	IDES AG	Frankfurt	DE
1010	wipro		
1101	WIPRO Technologies	hyderabad	IN
1819	emax technology	hyd	
1920	emax	amp	
2000	IDES UK	London	GB
2100	IDES Portugal	Lisbon	PT
2200	IDES France	Paris	FR
2300	IDES España	Barcelona	ES
2400	IDES Italia	Milano	IT
2500	IDES Netherlands	Rotterdam	NL
3000	IDES US INC	New York	US
4000	IDES Canada	Toronto	CA
4500	Canadian Company	Toronto	CA
5000	IDES Japan	Tokyo	JP
6000	IDES México, S.A. de C.V.	México DF	MX
7000	IDES Brazil	São Paulo	BR

SAPUSER basis INS

Reports

We Never Compromise In Quality, Would You?

If you double click on the company code a secondary list is displayed which displays the company code, customer number, Reconciliation Account.

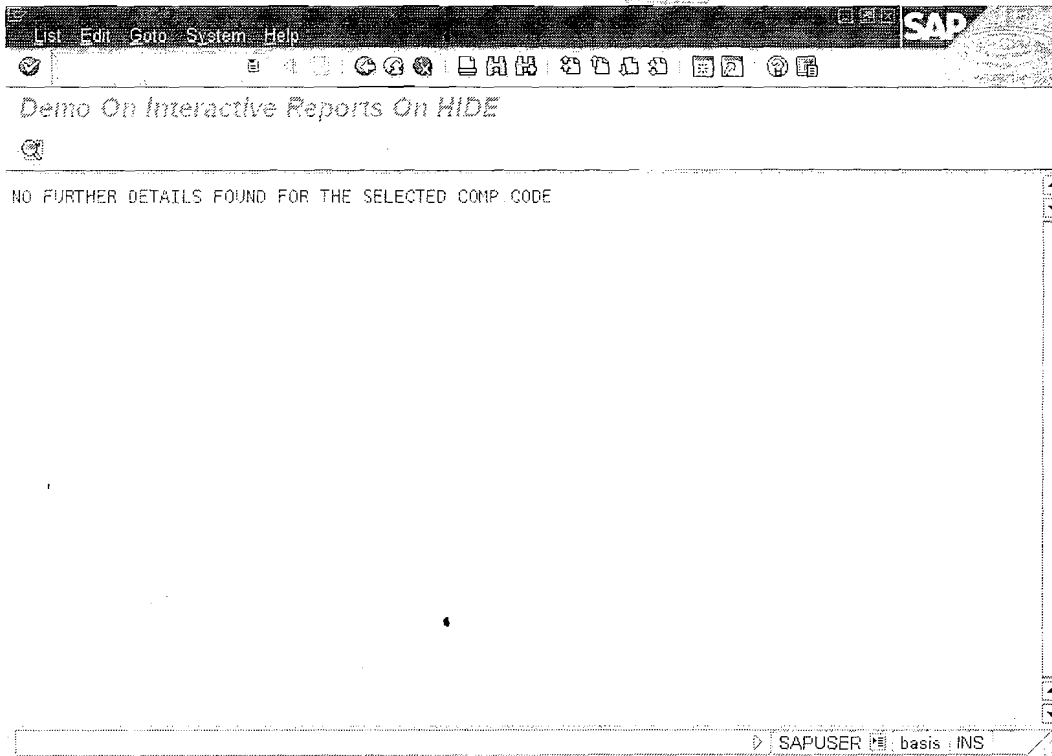


Demo On Interactive Reports On HIDE

Com. Code	Customer	Recon Acc
1000	224	140000
1000	1000	140000
1000	1001	140000
1000	1002	140000
1000	1003	140000
1000	1010	141000
1000	1012	140000
1000	1030	140000
1000	1031	140000
1000	1032	140000
1000	1033	140000
1000	1034	140000
1000	1050	140000
1000	1100	140000
1000	1110	140000
1000	1170	140000
1000	1171	140000
1000	1172	140000
1000	1173	140000

SAPUSER basis INS

If no further selected details of the company code are present then this screen is displayed.



Demo On Interactive Reports On HIDE

NO FURTHER DETAILS FOUND FOR THE SELECTED COMP CODE

SAPUSER basis INS

EXAMPLE PROGRAM2 USING HIDE :-

Reports
We Never Compromise In Quality, Would You?

* Program Heading

*AUTHOR : GANAPATI.ADIMULAM
*PROGRAM : ZDEMO_INTERACTIVE_HIDE_COMPLEX
*COPIED FROM : NA *
*PURPOSE : DISPLAY THE VENDOR DETAILS ON BASIC LIST
* AND PURCHASING HEADER DETAILS FOR THE
* SELECTED VENDOR IN THE NEXT SECONDARY
LIST*
* AND DISPLAY THE PURCHASING ITEMS FOR THE
* SLECTED PURCHASE DOC. IN NEXT LIST AND
* DISPLAY THE MATERIAL DETAILS FOR THE
* SELECTED MATERIAL IN NEXT LIST.
* REFERENCE DOCS : DEVWR0065

*MODIFICATION LOG :
*MOD-0001 : DETAILS ABOUT THE FIRST CHANGE IN THE PRO
*MOD-002 : DETAILS ABOUT THE 2ND CHANGE IN THE PROG
*TRANSPORT REQ.NO : C11DEV12354

REPORT **ZDEMO_INTERACTIVE_HIDE_COMPLEX** LINE-SIZE 150 NO
STANDARD PAGE HEADING LINE-COUNT 20 MESSAGE-ID ZDEMO.

TYPES : BEGIN OF TY_LFA1,
LIFNR TYPE LIFNR, "VENDOR NUMBER.
NAME1 TYPE NAME1, "NAME.
LAND1 TYPE LAND1, "COUNTRY KEY.
ORT01 TYPE ORT01, "CITY.
ANRED TYPE ANRED, "TITLE.
END OF TY_LFA1.

DATA : IT_LFA1 TYPE TABLE OF TY_LFA1,
WA_LFA1 TYPE TY_LFA1.

TYPES : BEGIN OF TY_EKKO,
LIFNR TYPE LIFNR, "VENDOR NUMBER.
EBELN TYPE EBELN, "PUCHASE DOCUMENT NUMBER.
BSTYP TYPE EBSTYP, "PURCHASING DOCUMENT CATEGORY
EKORG TYPE EKORG, "PURCHASING ORGANIZATION.
END OF TY_EKKO.

DATA : IT_EKKO TYPE STANDARD TABLE OF TY_EKKO,
WA_EKKO TYPE TY_EKKO .

Reports
We Never Compromise In Quality, Would You?

TYPES : BEGIN OF TY_EKPO,
EBELN TYPE EBELN, "PURCHASING DOCUMENT NUMBER.
EBELP TYPE EBELP, "ITEM NO OF PURCHASING NUMBER
MATNR TYPE MATNR, "MATERIAL NUMBER.
MENGE TYPE BSTMG, "PURCHASE ORDER QUANTITY.
NETPR TYPE BPREI, "NET PRICE IN PURCHASE DOCUMENT.
END OF TY_EKPO.

DATA : IT_EKPO TYPE STANDARD TABLE OF TY_EKPO,
WA_EKPO TYPE TY_EKPO .

TYPES : BEGIN OF TY_MAKT,
MATNR TYPE MATNR, "MATERIAL NUMBER.
MAKTX TYPE MAKTX, "MATERIAL DESCRIPTION.
END OF TY_MAKT.

DATA IT_MAKT TYPE STANDARD TABLE OF TY_MAKT.
DATA WA_MAKT TYPE TY_MAKT.

***** SELECTION-SCREEN *****

SELECTION-SCREEN BEGIN OF BLOCK SREE WITH FRAME TITLE
TEXT-001 .

DATA V_LIFNR TYPE LIFNR.
SELECT-OPTIONS : S_LIFNR FOR V_LIFNR.
SELECTION-SCREEN END OF BLOCK SREE.

***** AT SELECTION-SCREEN *****

AT SELECTION-SCREEN.
SELECT SINGLE LIFNR INTO S_LIFNR
FROM LFA1
WHERE LIFNR IN S_LIFNR.
IF SY-SUBRC NE 0.
MESSAGE E000. "NOT VALID VENDOR(S).
ENDIF.

* START-OF-SELECTION *

START-OF-SELECTION.

SELECT LIFNR
NAME1
LAND1
ORT01
ANRED INTO TABLE IT_LFA1
FROM LFA1

WHERE LIFNR IN S_LIFNR.

```
*****
*           END-OF-SELECTION.           *
*****
END-OF-SELECTION.
```

```
*DISPLAY VENDOR DETAILS BASIC LIST
LOOP AT IT_LFA1 INTO WA_LFA1.
  WRITE : /3 WA_LFA1-LIFNR,
         15 WA_LFA1-NAME1,
         52 WA_LFA1-LAND1,
         62 WA_LFA1-ORT01,
         95 WA_LFA1-ANRED.
```

```
*HIDE THE REQUIRED TO QUERRY THE NEXT LEVEL DATA
  HIDE : WA_LFA1-LIFNR.
  CLEAR : WA_LFA1.
  ENDLLOOP.
```

```
*****
*           TOP-OF-PAGE.           *
*****
```

```
TOP-OF-PAGE.
FORMAT COLOR 3.
WRITE : /3 'VENDOR DETAILS FROM LFA1 TABLE '.
FORMAT COLOR OFF.
ULINE.
WRITE : /3 'V.NUMBER',15 'NMAE',52 'COUNTRY',62 'CITY', 95 'TITLE'.
ULINE.
```

```
*****
*           AT LINE-SELECTION.       *
*****
```

```
AT LINE-SELECTION.
CASE SY-LSIND.
  WHEN 1.
* DISPLAY PUR.DOC.NO IN 1ST SECONDRY LIST
  SELECT LIFNR "VENDOR
         EBELN "PUR.DOC
         BSTYP "DOC.TYPE
         EKORG "PUR.DOC
  INTO TABLE IT_EKKO
  FROM EKKO
  WHERE LIFNR = WA_LFA1-LIFNR.
  IF NOT IT_EKKO IS INITIAL.
```

Reports

We Never Compromise In Quality, Would You?

```
LOOP AT IT_EKKO INTO WA_EKKO.
  AT FIRST.
    FORMAT COLOR 4.
    WRITE : /3 'PURCHASE DOCUMENT NUMBER DETAILS FROM
EKKO'.
    FORMAT COLOR OFF.
    ULINE.
    WRITE : /3 'VNUMBER', 15 'PDOC.NO',30 'DOC.CATGRY',
      45 'PO.ORG'.
    ULINE.
  ENDAT.
  WRITE : /3 WA_EKKO-LIFNR,
    15 WA_EKKO-EBELN,
    30 WA_EKKO-BSTYP,
    45 WA_EKKO-EKORG.
```

*HIDE EBELN TO QUERY NEXT LEVEL DATA FROM EKPO

```
HIDE : WA_EKKO-EBELN.
```

```
CLEAR : WA_EKKO.
```

```
ENDLOOP.
```

```
ELSE.
```

```
WRITE : /3 'NO RECORD(S) FOUND FOR THE SELECTED VENDOR'.
```

```
ENDIF.
```

WHEN 2.

* DISPLAY LINE ITEM IN 2ND SECONDRY LIST

```
SELECT EBELN
```

```
EBELP
```

```
MATNR
```

```
MENGE
```

```
NETPR
```

```
INTO TABLE IT_EKPO
```

```
FROM EKPO
```

```
WHERE EBELN = WA_EKKO-EBELN.
```

```
"WHICH WAS ALREADY HIDDEN AT PREVIOUS LIST
```

```
IF NOT IT_EKPO IS INITIAL.
```

```
LOOP AT IT_EKPO INTO WA_EKPO.
```

```
AT FIRST.
```

```
FORMAT COLOR 5.
```

```
WRITE : /3 'ITEM NO OF PURCHASE DOCUMENT FROM EKPO
TABLE'.
```

```
FORMAT COLOR OFF.
```

```
ULINE.
```

```
WRITE : /3 'PU.DOC.NO', 15 'ITEM.NO',25 'MATERIAL.NO',
```

```
45 'QUANTITY', 65 'NET.PRICE'.
```

```
ULINE.
```

Reports

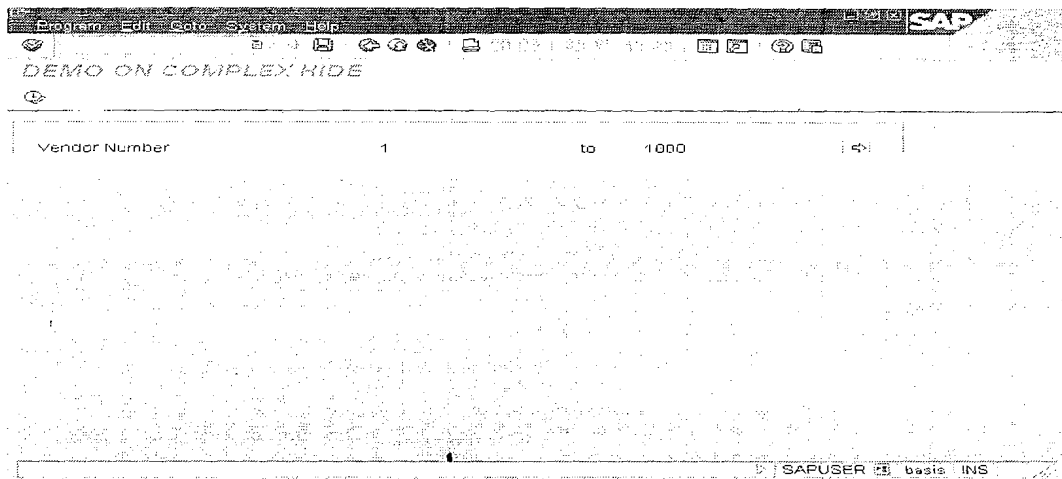
We Never Compromise In Quality, Would You?

```
ENDAT.
WRITE : /3 WA_EKPO-EBELN,
      15 WA_EKPO-EBELP,
      25 WA_EKPO-MATNR,
      45 WA_EKPO-MENGE LEFT-JUSTIFIED,
      65 WA_EKPO-NETPR LEFT-JUSTIFIED.
*HIDE THE MATERIAL NUMBER, TO QUERRY THE MORE ABOUT
MATERIALS
  HIDE : WA_EKPO-MATNR.
  CLEAR : WA_EKPO.
  ENDLOOP.
ELSE.
  WRITE : /3 'NO RECORD(S) FOUND FOR THE SELECTED RECORD'.
  ENDIF.
WHEN 3.
* DISPLAY MATERIALS DETAILS IN 3RD SECONDRY LIST
  SELECT MATNR
    MAKTX
  INTO TABLE IT_MAKT
  FROM MAKT
  WHERE MATNR = WA_EKPO-MATNR AND
        SPRAS = SY-LANGU.
  IF NOT IT_MAKT IS INITIAL.
    LOOP AT IT_MAKT INTO WA_MAKT.
    AT FIRST.
      FORMAT COLOR 1.
      WRITE : /3 'MATERIAL,NO & DESCRIPTION DETAILS'.
      FORMAT COLOR OFF.
      ULINE.
      WRITE : /3 'MATERIAL.NO',15 'MAT.DESC'.
      ULINE.
    ENDAT.
    WRITE : /3 WA_MAKT-MATNR,
          15 WA_MAKT-MAKTX.
    CLEAR WA_MAKT.
  ENDLOOP.
ELSE.
  WRITE : /3 'NO RECORD FOUND FOR THE SELECTED RECORD'.
  ENDIF.
ENDCASE.
```

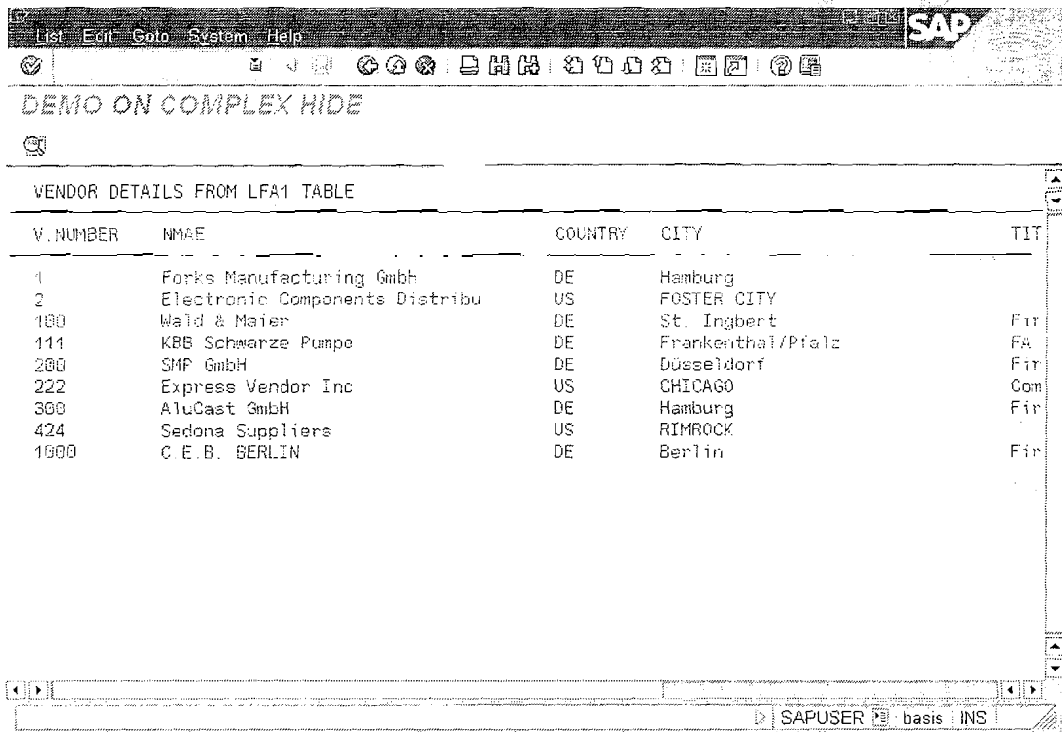

Reports

We Never Compromise In Quality, Would You?

INPUT:-



OUTPUT:-



Double Click on Vendor Number 1000

Reports
We Never Compromise In Quality, Would You?

DEMO ON COMPLEX HIDE

VNUMBER	PDCC. NO	DOC. CATGRY	PO. ORG
1000	4500012884	F	1000
1000	4500012885	F	1000
1000	4500012886	F	1000
1000	4500000000	K	1000
1000	4500000001	K	1000
1000	4500000002	K	1000
1000	4500000003	K	1000
1000	4500000022	K	1000
1000	450004865	F	1000
1000	450004968	F	1000
1000	450005139	F	1000
1000	450005148	F	1000
1000	450005354	F	1000
1000	450005851	F	1000
1000	450005874	F	1000
1000	450006126	F	1000
1000	450006130	F	1000

SAPUSER basis INS

Double Click on the Purchase Document Number.

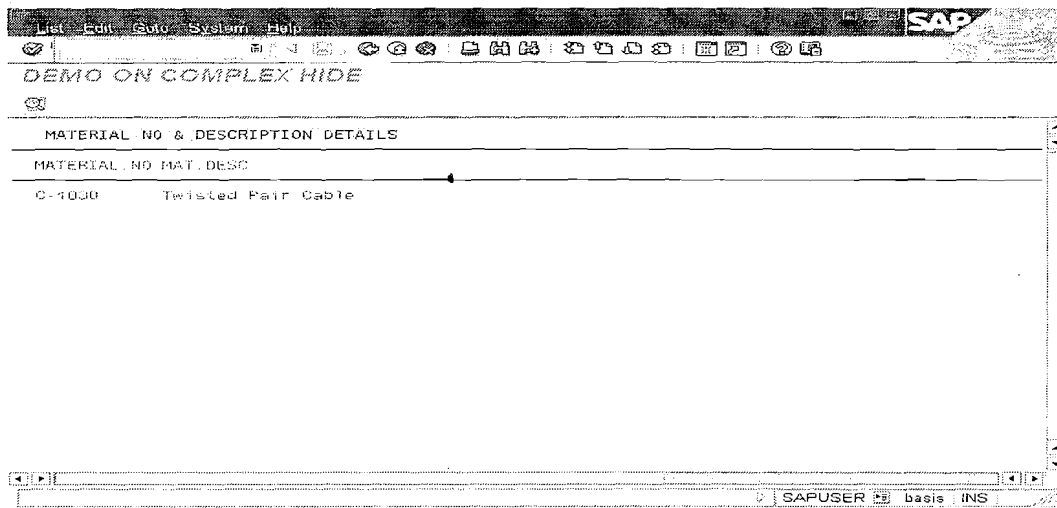
DEMO ON COMPLEX HIDE

PU. DOC. NO	ITEM NO	MATERIAL. NO	QUANTITY	NET. PRICE
450004865	00010	C-1030	2,000,000	10,00
450004865	00020	C-1031	2,000,000	11,00
450004865	00030	C-1032	2,000,000	12,00

SAPUSER basis INS

Double click on the Material Number.

Reports
We Never Compromise In Quality, Would You?



WORKING WITH GET CURSOR :

Reading Lists at the Cursor Position :

Note : When the User is Interested with Selected Field Details, i.e when the User Interaction depends on the Selected Field Name and Field Value , Both the above techniques : SY-LISEL and HIDE are not enough because they Can return only the Contents but the Selected Field Names. So in this Case it is Mandatory to go for **GET CURSOR** Technique, Which gives both Selected Field Name and Value.

Syntax :

GET CURSOR FIELD <V_FNAM> VALUE <V_FVAL>.

Note:

This statement transfers the name of the field on which the cursor is positioned during a user action into the variable <V_FNAM> and Field Value into <V_FVAL>.

Note : If the cursor is on a field, the system sets SY-SUBRC to 0, otherwise to 4.

EXAMPLE PROGRAM ON GET CURSOR:

* Program Heading

*AUTHOR : GANAPATI.ADIMULAM
*PROGRAM : ZDEMO_INTERACTIVE_GET_CURSOR
*COPIED FROM : NA
*PURPOSE : DISPLAY THE COMPANY DETAILS ON BASIC LIST
* AND CUSTOMER DETAILS AND Reconciliation
* Account
* : IS DEVELOPED USING GET CURSOR TECHNIQUE
*REFERENCE DOCS : DEVWR0065

*MODIFICATION LOG :

*MOD-0001 : DETAILS ABOUT THE FIRST CHANGE IN THE PROG
*MOD-002 : DETAILS ABOUT THE 2ND CHANGE IN THE PROG
*TRANSPORT REQ.NO : C11DEV12354

REPORT **ZDEMO_INTERACTIVE_GET_CURS**ROR.

TYPES : BEGIN OF TY_T001,
BUKRS TYPE BUKRS, "COMPANY CODE
BUTXT TYPE BUTXT, "NAME OF THE COMPANY
ORT01 TYPE ORT01, "CITY
LAND1 TYPE LAND1, "COUNTRY KEY
WAERS TYPE WAERS, "CURRENCY KEY
END OF TY_T001.

DATA : WA_T001 TYPE TY_T001,
IT_T001 TYPE STANDARD TABLE OF TY_T001.

DATA : V_BUKRS TYPE BUKRS,
V_FNAM(15),
V_FVAL TYPE BUKRS.

*CUSTOMER MASTER DATA

TYPES : BEGIN OF TY_CUSTOMER,
KUNNR TYPE KUNNR, "CUSTOMER NO
BUKRS TYPE BUKRS, "COMPANY CODE
AKONT TYPE AKONT, "RECONCILIATION ACCOUNT
END OF TY_CUSTOMER.

DATA : WA_CUSTOMER TYPE TY_CUSTOMER,
IT_CUSTOMER TYPE STANDARD TABLE OF TY_CUSTOMER.

*DESIGNING THE SELECTION SCREEN
SELECT-OPTIONS : S_BUKRS FOR V_BUKRS.

Reports
We Never Compromise In Quality, Would You?

```
***** START-OF-SELECTION. *****
START-OF-SELECTION.
SELECT  BUKRS "COMPANY CODE
        BUTXT "NAME OF THE COMPANY
        ORT01 "CITY
        LAND1 "COUNTRY KEY
        WAERS "CURRENCY KEY
INTO TABLE IT_T001
FROM T001
WHERE BUKRS IN S_BUKRS.
```

```
***** END-OF-SELECTION. *****
END-OF-SELECTION.
```

```
IF NOT IT_T001 IS INITIAL.
WRITE :/5 'COM.CODE', 15 'TEXT', 45 'CITY', 70 'COUNTRY'.
ULINE.
LOOP AT IT_T001 INTO WA_T001.
WRITE :/5 WA_T001-BUKRS,
        15 WA_T001-BUTXT,
        45 WA_T001-ORT01,
        70 WA_T001-LAND1.
CLEAR WA_T001.
ENDLOOP.
ENDIF.
```

```
***** AT LINE-SELECTION. *****
AT LINE-SELECTION.
```

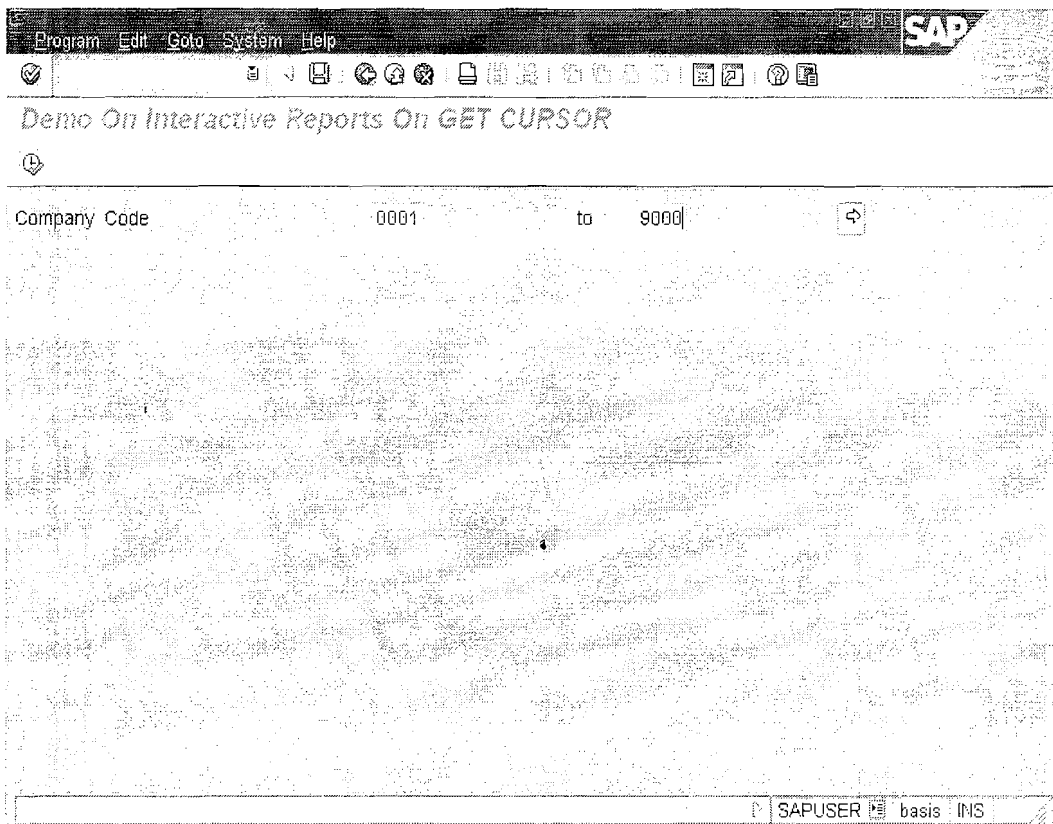
```
GET CURSOR FIELD V_FNAM VALUE V_FVAL.
CASE SY-LSIND.
WHEN 1.
IF V_FNAM = 'WA_T001-BUKRS'.
*RETRIEVE ALL THE CUSTOMERS FOR THE SELECTED COMPANY CODE
*INNER JOIN
SELECT KUNNR "CUSTOMER
        BUKRS "COMPANY CODE
        AKONT "RECON.ACC
INTO TABLE IT_CUSTOMER
FROM KNB1
WHERE BUKRS = V_FVAL.
*DISPLAY THE DATA
IF NOT IT_CUSTOMER IS INITIAL.
WRITE :/5 'COM.CODE', 15 'CUSTOMER', 30 'RECON ACC'.
ULINE.
```

Reports

We Never Compromise In Quality, Would You?

```
LOOP AT IT_CUSTOMER INTO WA_CUSTOMER.  
  WRITE : /5 WA_CUSTOMER-BUKRS, "COMPANY CODE  
    15 WA_CUSTOMER-KUNNR, "CUSTOMER  
  
    30 WA_CUSTOMER-AKONT. "RECONCILIATION ACCOUNT  
  CLEAR WA_CUSTOMER.  
ENDLOOP.  
ELSE.  
  WRITE : / 'NO FURTHER DETAILS FOUND FOR THE SELECTED  
COMP.CODE'.  
ENDIF.  
ENDIF.  
ENDCASE.
```

OUTPUT :-EXECUTE THE PROGRAM



Reports

We Never Compromise In Quality, Would You?

Demo On Interactive Reports On GET CURSOR

Com. Code	Text	City	Country
0001	SAP A.G.	Walldorf	DE
1000	IDES AG	Frankfurt	DE
1010	wipro		
1101	WIPRO Technologies	hyderabad	IN
1018	emax technology	hyd	
1020	emax	amp	
2000	IDES UK	London	GB
2100	IDES Portugal	Lisbon	PT
2200	IDES France	Paris	FR
2300	IDES España	Barcelona	ES
2400	IDES Italia	Milano	IT
2500	IDES Netherlands	Rotterdam	NL
3000	IDES US INC	New York	US
4000	IDES Canada	Toronto	CA
4500	Canadian Company	Toronto	CA
5000	IDES Japan	Tokyo	JP
6000	IDES México, S.A. de C.V.	México DF	MX
7000	IDES Brazil	São Paulo	BR

SAPUSER basis INS

If you double click on the company code a secondary list is displayed which displays the company code, customer number, Reconciliation Account.

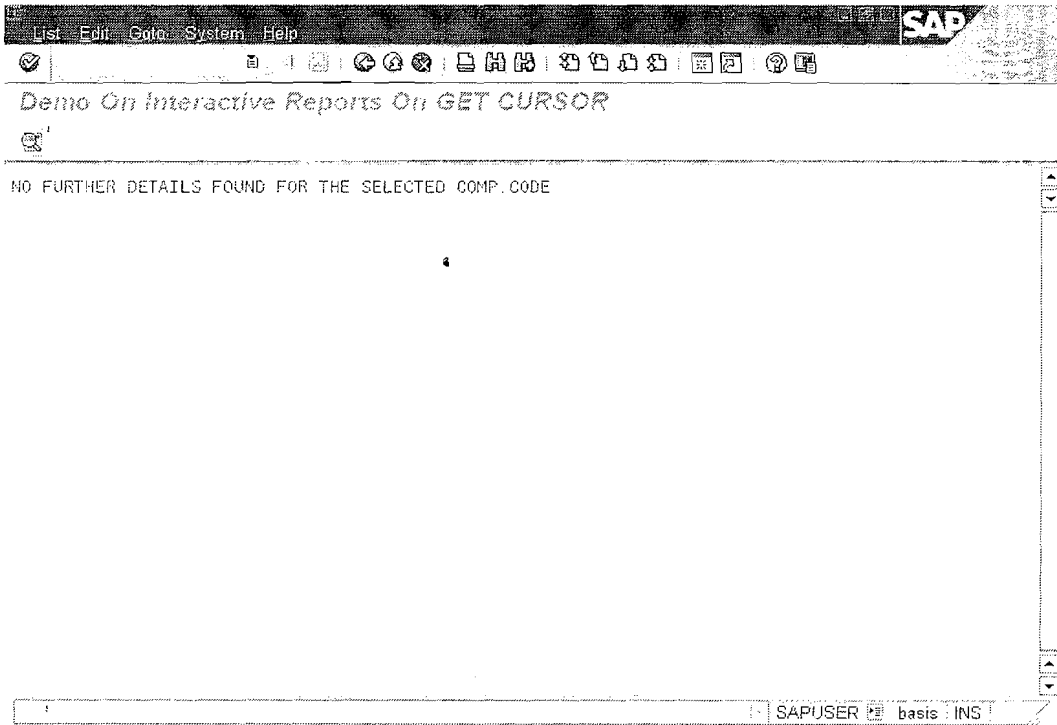
Demo On Interactive Reports On GET CURSOR

Com Code	Customer	Recon Acc
3000	224	140000
3000	255	140000
3000	256	140000
3000	257	140000
3000	258	140000
3000	259	140000
3000	260	140000
3000	261	140000
3000	262	140000
3000	263	140000
3000	264	140000
3000	265	140000
3000	266	140000
3000	267	140000
3000	268	140000
3000	269	140000
3000	270	140000
3000	271	140000
3000	272	140000
3000	273	140000

SAPUSER basis INS

If no further selected details of the company code are present then this screen is displayed.

Reports
We Never Compromise In Quality, Would You?



EXAMPLE PROGRAM2 USING GET CURSOR:-

Note: In this Program, We Use Conversion Exit.

Conversion Routines are Maintained at Domain Level of the Data Elements And For those fields which refer those Data Elements, the External(Output) Format and Internal(Database) Format Differs. So that Before we use the Selected Data From the Output list, it has to be Converted to Internal Format then Select the Next Level Data.

CALL FUNCTION 'CONVERSION_EXIT_ALPHA_INPUT'

EXPORTING

INPUT = V_FVAL_LIFNR "Internal Format

IMPORTING

OUTPUT = V_FVAL_LIFNR. "Output Format

```
*****
*          Program Heading
*****
*AUTHOR      : GANAPATI.ADIMULAM
*PROGRAM     : ZDEMO_GET_CURSOR_COMPLEX
*COPIED FROM : NA
*PURPOSE     : DISPLAY THE VENDOR DETAILS ON BASIC LIST
*             AND PURCHASING HEADER DETAILS FOR THE
*             SELECTED VENDOR IN THE NEXT SECONDARY
LIST*
*             AND DISPLAY THE PURCHASING ITEMS FOR THE
*             SLECTED PURCHASE DOC. IN NEXT LIST AND
*             DISPLAY THE MATERIAL DETAILS FOR THE
*             SELECTED MATERIAL IN NEXT LIST.
*             : IS DEVELOPED USING GET CURSOR TECHNIQUE
*REFERENCE DOCS : DEVWR0065
*****
*MODIFICATION LOG :
*MOD-0001      : DETAILS ABOUT THE FIRST CHANGE IN THE PROG
*MOD-002      : DETAILS ABOUT THE 2ND CHANGE IN THE PROG
*TRANSPORT REQ.NO : C11DEV12354
*****
REPORT ZDEMO_GET_CURSOR_COMPLEX LINE-SIZE 150 NO
STANDARD PAGE HEADING LINE-COUNT 20 MESSAGE-ID ZDEMO.

TYPES : BEGIN OF TY_LFA1,
        LIFNR TYPE LIFNR, "VENDOR NUMBER.
        NAME1 TYPE NAME1, "NAME.
        LAND1 TYPE LAND1, "COUNTRY KEY.
        ORT01 TYPE ORT01, "CITY.
        ANRED TYPE ANRED, "TITLE.
END OF TY_LFA1.
```

Reports
We Never Compromise In Quality, Would You?

DATA : IT_LFA1 TYPE TABLE OF WA_LFA1,
WA_LFA1 TYPE TY_LFA1.

TYPES : BEGIN OF TY_EKKO,
LIFNR TYPE LIFNR, "VENDOR NUMBER.
EBELN TYPE EBELN, "PURCHASE DOCUMENT NUMBER.
BSTYP TYPE EBSTYP,
"PURCHASING DOCUMENT CATEGORY
EKORG TYPE EKORG, "PURCHASING ORGANIZATION.
END OF TY_EKKO.

DATA : IT_EKKO TYPE STANDARD TABLE OF TY_EKKO,
WA_EKKO TYPE TY_EKKO.

TYPES: BEGIN OF TY_EKPO,
EBELN TYPE EBELN, "PURCH DOCUMENT NUMBER.
EBELP TYPE EBELP, "ITEM NO OF PURCHASING NUMBER
MATNR TYPE MATNR, "MATERIAL NUMBER.
MENGE TYPE BSTMG, "PURCHASE ORDER QUANTITY.
NETPR TYPE BPRI, "NET PRICE
END OF TY_EKPO.

DATA : WA_EKPO TYPE TY_EKPO,
IT_EKPO TYPE STANDARD TABLE OF TY_EKPO.

TYPES : BEGIN OF TY_MAKT,
MATNR TYPE MATNR, "MATERIAL NUMBER.
MAKTX TYPE MAKTX, "MATERIAL DESCRIPTION.
END OF TY_MAKT.

DATA IT_MAKT TYPE STANDARD TABLE OF TY_MAKT.

DATA : V_FNAM(15) TYPE C,
V_FVAL_LIFNR TYPE LIFNR,
V_FVAL_EBELN TYPE EBELN,
V_FVAL_MATNR TYPE MATNR.

***** SELECTION-SCREEN *****
SELECTION-SCREEN BEGIN OF BLOCK SREE WITH FRAME TITLE
TEXT-001 .

DATA V_LIFNR TYPE LIFNR.
SELECT-OPTIONS : S_LIFNR FOR V_LIFNR.
SELECTION-SCREEN END OF BLOCK SREE.

***** AT SELECTION-SCREEN *****
AT SELECTION-SCREEN.
SELECT SINGLE LIFNR INTO S_LIFNR
FROM LFA1

Reports
We Never Compromise In Quality, Would You?

```
WHERE LIFNR IN S_LIFNR.
IF SY-SUBRC NE 0.
  MESSAGE E000. "NOT VALID VENDOR(S).
ENDIF.
***** START-OF-SELECTION *****
START-OF-SELECTION.
SELECT LIFNR
  NAME1
  LAND1
  ORT01
  ANRED INTO TABLE IT_LFA1
FROM LFA1
WHERE LIFNR IN S_LIFNR.

***** END-OF-SELECTION *****
END-OF-SELECTION.
*DISPLAY VENDOR DETAILS BASIC LIST
LOOP AT IT_LFA1 INTO WA_LFA1.
  WRITE :/3 WA_LFA1-LIFNR,
    15 WA_LFA1-NAME1,
    52 WA_LFA1-LAND1,
    62 WA_LFA1-ORT01,
    95 WA_LFA1-ANRED.
  CLEAR : WA_LFA1.
ENDLOOP.
***** TOP-OF-PAGE *****
TOP-OF-PAGE.
FORMAT COLOR 3.
WRITE :/3 'VENDOR DETAILS FROM LFA1 TABLE'.
FORMAT COLOR OFF.
ULINE.
WRITE :/3 'V.NUMBER',15 'NMAE',52 'COUNTRY',62 'CITY', 95 'TITLE'.
ULINE.
***** AT LINE-SELECTION*****
AT LINE-SELECTION.
CASE SY-LSIND.
  WHEN 1.
    GET CURSOR FIELD V_FNAM VALUE V_FVAL_LIFNR.
    * CONVERTS FROM EXTERNAL TO INTERNAL FORMAT
    CALL FUNCTION 'CONVERSION_EXIT_ALPHA_INPUT'
      EXPORTING
        INPUT = V_FVAL_LIFNR
      IMPORTING
        OUTPUT = V_FVAL_LIFNR.
    IF V_FNAM = 'WA_LFA1-LIFNR'.
      * DISPLAY PUR.DOC.NO IN 1ST SECONDRY LIST
```

Which Converts External(Display) Format to Internal(Database Format).

Reports

We Never Compromise In Quality, Would You?

```
SELECT LIFNR "VENDOR
      EBELN "PUR.DOC
      BSTYP "DOC.TYPE
      EKORG "PUR.DOC
INTO TABLE IT_EKKO
FROM EKKO
WHERE LIFNR = V_FVAL_LIFNR.
IF NOT IT_EKKO IS INITIAL.
  LOOP AT IT_EKKO INTO WA_EKKO.
    AT FIRST.
      FORMAT COLOR .
      WRITE : /3 'PURCHASE DOCUMENT NUMBER DETAILS FROM
EKKO'.
      FORMAT COLOR OFF.
      ULINE.
      WRITE : /3 'VNUMBER', 15 'PDOC.NO',30 'DOC.CATGRY',
        45 'PO.ORG'.
      ULINE.
      ENDAT.
      WRITE :/3 WA_EKKO-LIFNR,
        15 WA_EKKO-EBELN,
        30 WA_EKKO-BSTYP,
        45 WA_EKKO-EKORG.
      CLEAR : WA_EKKO.
      ENDLOOP.
    ELSE.
      WRITE : /3 'NO RECORD(S) FOUND FOR THE SELECTED VENDOR'.
      ENDIF.
    ENDIF . "WA_LFA1-LIFNR
WHEN 2.
  CLEAR V_FNAM.
  GET CURSOR FIELD V_FNAM VALUE V_FVAL_EBELN.
  IF V_FNAM = 'WA_EKKO-EBELN'.
* CONVERTS FROM EXTERNAL TO INTERNAL FORMAT
  CALL FUNCTION 'CONVERSION_EXIT_ALPHA_INPUT'
    EXPORTING
      INPUT = V_FVAL_EBELN
    IMPORTING
      OUTPUT = V_FVAL_EBELN.
* DISPLAY LINE ITEM IN 2ND SECONDRY
  SELECT EBELN
    EBELP
    MATNR
    MENGE
    NETPR
  INTO TABLE IT_EKPO
```

Which Converts External(Display)
Format to Internal(Database Format).

Reports

We Never Compromise In Quality, Would You?

```
FROM EKPO
WHERE EBELN = V_FVAL_EBELN.
IF NOT IT_EKPO IS INITIAL.
LOOP AT IT_EKPO INTO WA_EKPO.
  AT FIRST.
    FORMAT COLOR 5.
    WRITE : /3 'ITEM NO OF PURCHASE DOCUMENT FROM EKPO
TABLE'.
    FORMAT COLOR OFF.
    ULINE.
    WRITE : /3 'PU.DOC.NO', 15 'ITEM.NO', 25 'MATERIAL.NO',
      45 'QUANTITY', 65 'NET.PRICE'.
    ULINE.
    ENDAT.
    WRITE : /3 WA_EKPO-EBELN,
      15 WA_EKPO-EBELP,
      25 WA_EKPO-MATNR,
      45 WA_EKPO-MENGE LEFT-JUSTIFIED,
      65 WA_EKPO-NETPR LEFT-JUSTIFIED.
    CLEAR : WA_EKPO.
  ENDLOOP.
ELSE.
  WRITE : /3 'NO RECORD(S) FOUND FOR THE SELECTED RECORD'.
ENDIF.
ENDIF. "WA_EKKO-EBELN
WHEN 3.
* DISPLAY MATERIALS DETAILS IN 3RD SECONDRY LIST
CLEAR V_FNAM.
GET CURSOR FIELD V_FNAM VALUE V_FVAL_MATNR.
IF V_FNAM = 'WA_EKPO-MATNR'.
* CONVERTS FROM EXTERNAL TO INTERNAL FORMAT
CALL FUNCTION 'CONVERSION_EXIT_ALPHA_INPUT'
  EXPORTING
    INPUT = V_FVAL_MATNR
  IMPORTING
    OUTPUT = V_FVAL_MATNR.
SELECT MATNR
  MAKTX
  INTO TABLE IT_MAKT
  FROM MAKT
  WHERE MATNR = V_FVAL_MATNR AND
    SPRAS = SY-LANGU.
IF NOT IT_MAKT IS INITIAL.
LOOP AT IT_MAKT INTO WA_MAKT.
  AT FIRST.
    FORMAT COLOR 1.
```

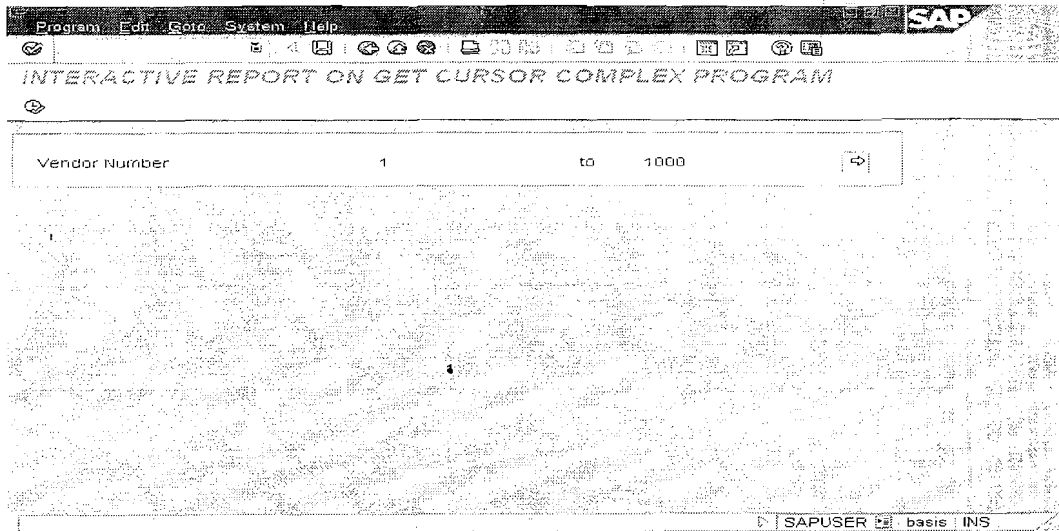
Which Converts External(Display)
Format to Internal(Database Format).

Reports

We Never Compromise In Quality, Would You?

```
WRITE : /3 ' MATERIAL NO & DESCRIPTION DETAILS'.  
FORMAT COLOR OFF.  
ULINE.  
WRITE : /3 ' MATERIAL.NO',15 ' MAT.DESC'.  
ULINE.  
ENDAT.
```

```
WRITE : /3  WA_MAKT-MATNR,  
          15 WA_MAKT-MAKTX.  
CLEAR WA_MAKT.  
ENDLOOP.  
ELSE.  
WRITE : /3 ' NO RECORD FOUND FOR THE SELECTED RECORD'.  
ENDIF.  
ENDIF . "WA_EKPO-MATNR  
ENDCASE.
```



Reports

We Never Compromise In Quality, Would You?

List Edit Goto System Help SAP

INTERACTIVE REPORT ON GET CURSOR COMPLEX PROGRAM

VENDOR DETAILS FROM LFA1 TABLE

V. NUMBER	NMAE	COUNTRY	CITY	TIT
1	Fork's Manufacturing GmbH	DE	Hamburg	
2	Electronic Components Distribu	US	FOSTER CITY	
100	Wald & Maier	DE	St. Ingbert	Fir
111	KBB Schwarze Pumpe	DE	Frankenthal/Pfalz	FA
260	SMP GmbH	DE	Düsseldorf	Fir
222	Express Vendor Inc	US	CHICAGO	Com
300	AluCast GmbH	DE	Hamburg	Fir
424	Sedona Suppliers	US	RIMROCK	
1000	C.E.B. BERLIN	DE	Berlin	Fir

SAPUSER basis INS

List Edit Goto System Help SAP

INTERACTIVE REPORT ON GET CURSOR COMPLEX PROGRAM

PURCHASE DOCUMENT NUMBER DETAILS FROM EKKO

VNUMBER	PDOC. NO	DOC. CATGRY	P0. ORG
1000	4500012884	F	1000
1000	4500012885	F	1000
1000	4500012886	F	1000
1000	4600000000	K	1000
1000	4600000001	K	1000
1000	4600000002	K	1000
1000	4600000003	K	1000
1000	4600000022	K	1000
1000	4500004865	F	1000
1000	4500004868	F	1000
1000	4500005139	F	1000
1000	4500005140	F	1000
1000	4500005354	F	1000
1000	4500005851	F	1000
1000	4500005874	F	1000
1000	4500006126	F	1000
1000	4500006130	F	1000

SAPUSER basis INS

Reports

We Never Compromise In Quality, Would You?

SAP

List Edit Goto System Help

INTERACTIVE REPORT ON GET CURSOR COMPLEX PROGRAM

ITEM NO OF PURCHASE DOCUMENT FROM EKPO TABLE

PU. DOC. NO	ITEM. NO	MATERIAL. NO	QUANTITY	NET. PRICE
4500004885	00010	C-1030	2.000,000	10,00
4500004885	00020	C-1031	2.000,000	11,00
4500004885	00030	C-1032	2.000,000	12,00

SAPUSER basis INS

SAP

List Edit Goto System Help

INTERACTIVE REPORT ON GET CURSOR COMPLEX PROGRAM

MATERIAL NO & DESCRIPTION DETAILS

MATERIAL. NO	MAT. DESC
C-1030	Twisted Pair Cable

SAPUSER basis INS

WORKING WITH EVENT AT USER-COMMAND:

DAY-5

EVENT AT USER-COMMAND Triggers When the User Interacts with Custom Function Keys (Buttons).

NOTE: Adding the Custom Function keys to the Output list is always through the Custom GUI Status and the Same has to be Created for the Program and attached to the required Output List Of the Program.

NOTE : Each Function Code is Identified Uniquely, and it is Collected into System Variable SY-UCOMM automatically, Each time the User Interacts with the Function Key.

Flow Of AT USER-COMMAND :

AT USER-COMMAND.

CASE SY-UCOMM.

WHEN 'FCODE1'.

--

--

WHEN 'FCODE2'.

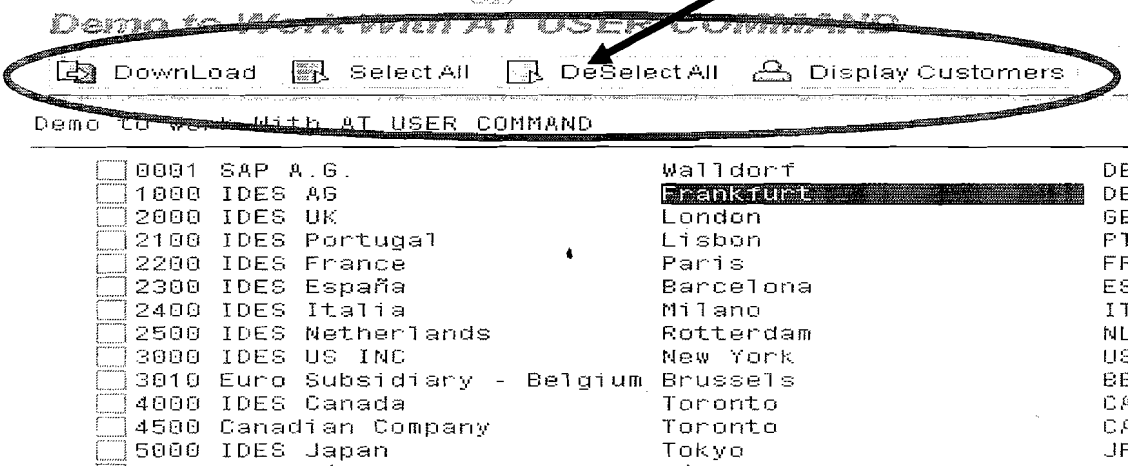
--

--

ENDCASE.

REQUIREMENT : DESIGN THE CUSTOM GUI STATUS WITH Function Keys Download, Select All, Deselect All On the Basic List.

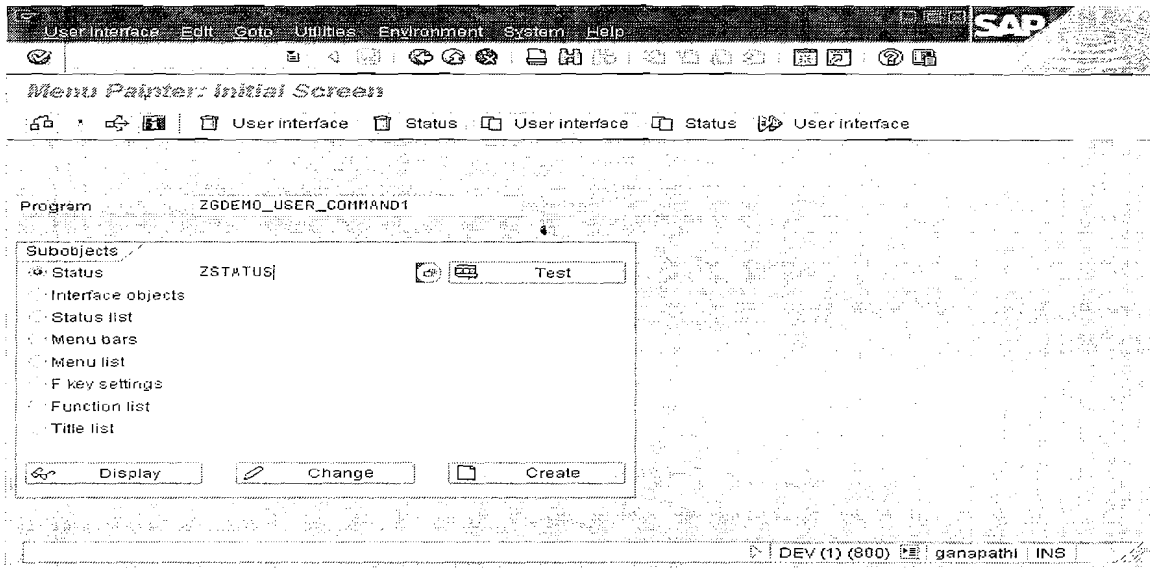
Example Screen :



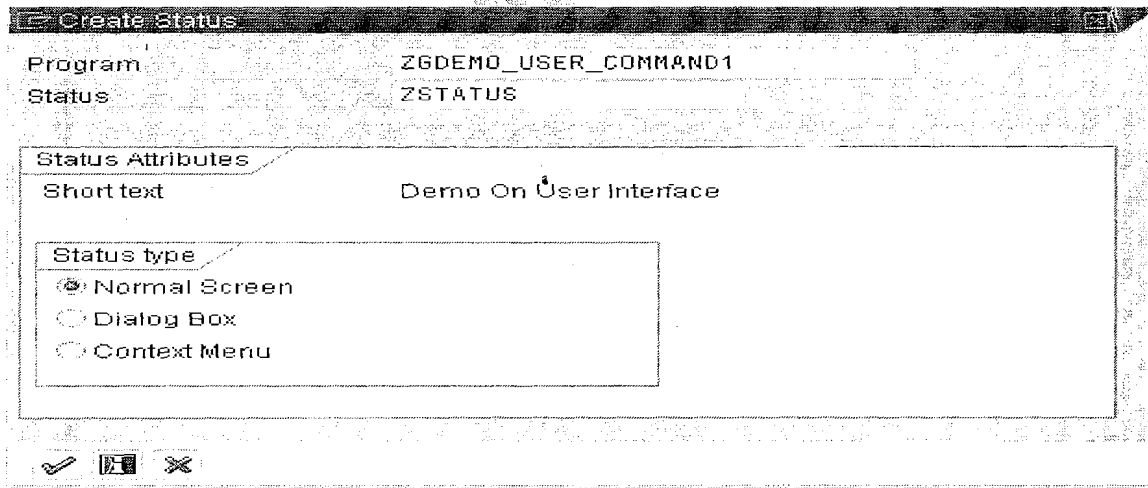
Steps to Create a custom GUI status

Define the push button **Export/Download**.

EXECUTE SE41 (MENU PAINTER)

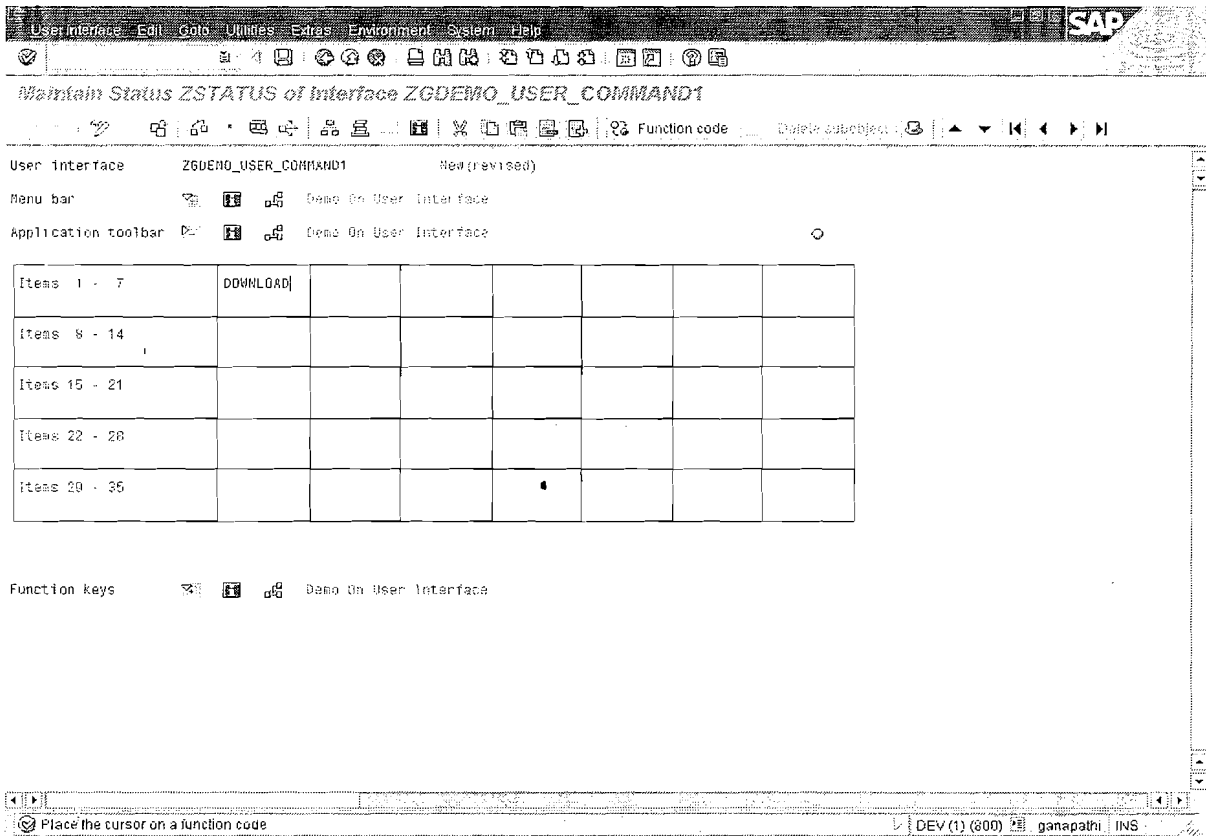


CREATE.



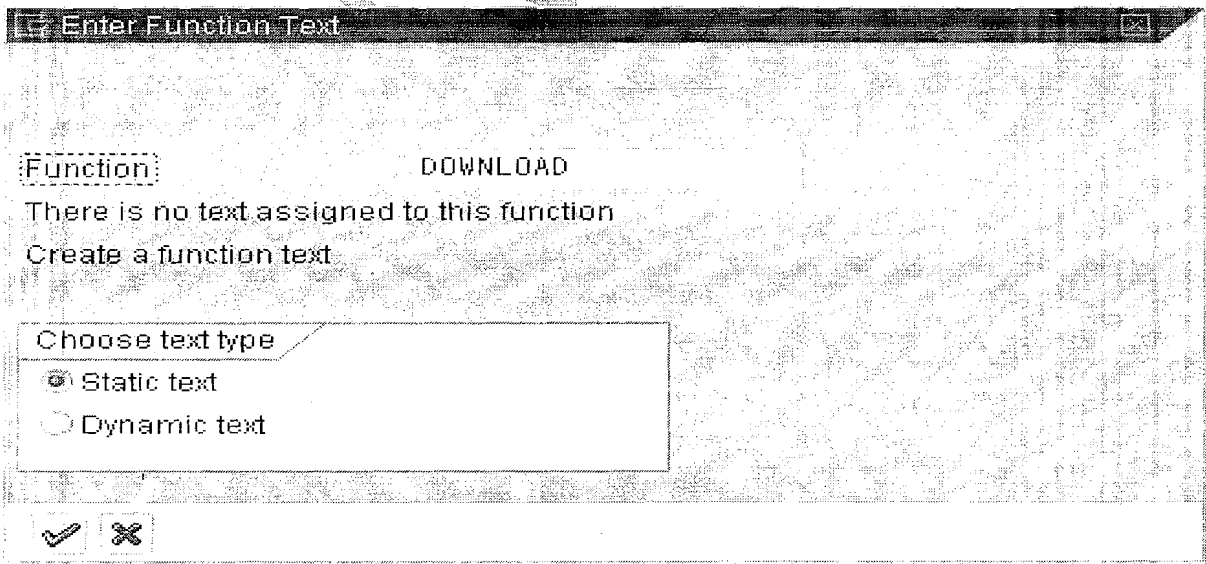
ENTER

Reports
We Never Compromise In Quality, Would You?



CLICK ON Application Toolbar & Enter the Function Code DOWNLOAD, Which is the Unique Identification for the Function Key.

Double Click On FCODE – DOWNLOAD



ENTER

Function code	DOWNLOAD
Functional type	Application function
Static function texts	
Function text	DownLoad
Icon name	ICON_EXPORT
Icon text	DownLoad
Info. text	Download to File
Fastpath	

Change text type

Provide Function Text , Info.text and Icon Name Details for Display Purpose.

ENTER

DOWNLOAD
DownLoad
is not assigned to a function key
Choose a function key

- F2
- F5
- F6
- F7
- F8
- F9
- Shift-F1
- **Shift-F2**
- Shift-F4
- Shift-F5
- Shift-F6
- Shift-F7
- Shift-F8
- Shift-F9
- Shift-Ctrl-0
- Shift-F11

Double Click On Some Shortcut Key.

Function Key DOWNLOAD Is Completed.

Note : Repeat the Same for other Function Codes : SALL(Select All) and DALLI(Deselect All).

The screenshot shows the SAP User Interface for the program ZGDEMO_USER_COMMAND1. The main area contains a table with the following data:

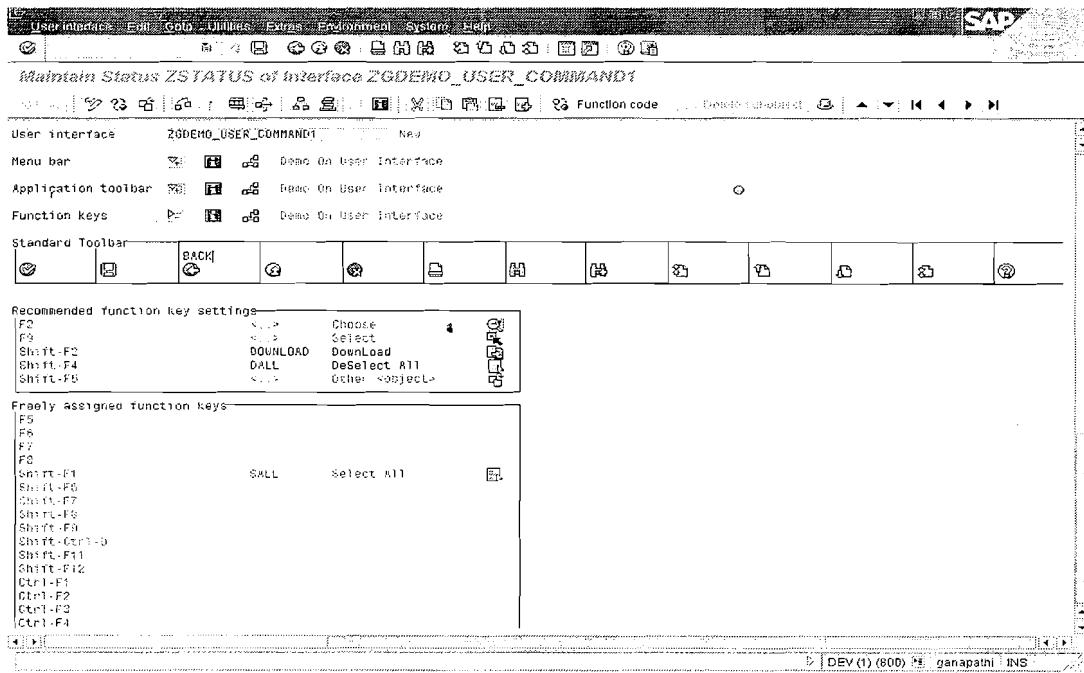
Items	DOWNLOAD	SALL	DALL					
1 - 7								
8 - 14								
15 - 21								
22 - 28								
29 - 35								

At the bottom of the screenshot, a status bar displays the message: "interface of program ZGDEMO_USER_COMMAND1 has been saved".

SAVE it.
Click On Function Keys from the Same Screen.

Reports

We Never Compromise In Quality, Would You?



Provide the Function Code as BACK For BACK and this function works automatically.

SAVE,CHECK and ACTIVATE .

Note : Creating the GUI STATUS for the Program is not enough, It has to be Linked to the Output List via SET PF-STATUS <Status Name> for the required Output List.

Reading Lines from Lists

All of the lists generated by a single program are stored internally in the system. You can therefore access any list in a program that was created for the same screen and that has not yet been deleted by returning to a lower list level. To read lines, use the statements READ LINE and **READ CURRENT LINE**.

To read a line from a list after an interactive list event, use the READ LINE statement:

```
READ LINE <lin> [INDEX <idx>]
[FIELD VALUE <f1> [INTO <g1>] ... <fn> [INTO <gn>]] .
```

```

*&-----*
*& Report ZGDEMO_USER_COMMAND1 *
*& *
*&-----*
    
```

REPORT ZGDEMO_USER_COMMAND1

```

TYPES : BEGIN OF TY_T001,
        BUKRS TYPE BUKRS, "COMPANY CODE
        BUTXT TYPE BUTXT, "Name of the company
        ORT01 TYPE ORT01, "CITY
        LAND1 TYPE LAND1, "COUNTRY KEY
        WAERS TYPE WAERS, "CURRENCY KEY
    END OF TY_T001.
    
```

```

TYPES : BEGIN OF TY_KNB1,
        BUKRS TYPE BUKRS, "CCODE
        KUNNR TYPE KUNNR, "Customer No
        ZTERM TYPE DZTERM, "Payment Terms
    END OF TY_KNB1.
    
```

```

DATA : IT_T001 TYPE STANDARD TABLE OF TY_T001,
        WA_T001 TYPE TY_T001.
    
```

```

DATA : IT_KNB1 TYPE STANDARD TABLE OF TY_KNB1,
        WA_KNB1 TYPE TY_KNB1.
    
```

```

DATA : IT_SELECTED_T001 TYPE STANDARD TABLE OF TY_T001.
DATA : V_BOX(1) TYPE C,
        V_LINES TYPE I,
        V_BUKRS TYPE BUKRS.
    
```

```

*DESIGNING THE SELECTION SCREEN
SELECT-OPTIONS : S_BUKRS FOR V_BUKRS.
    
```

```

*****
***
*          START-OF-SELECTION.          *
    
```

START-OF-SELECTION.

```
SELECT  BUKRS "COMPANY CODE
        BUTXT "Name of the company
        ORT01 "CITY
        LAND1 "COUNTRY KEY
        WAERS "CURRENCY KEY
INTO TABLE IT_T001
FROM T001
WHERE BUKRS IN S_BUKRS.
```

* END-OF-SELECTION. *

END-OF-SELECTION.

```
*SETTING USER DEFINED GUI STATUS
*HERE WE HAVE DOWNLOAD BUTTON
SET PF-STATUS 'ZSTATUS' .
```

```
IF NOT IT_T001 IS INITIAL.
  LOOP AT IT_T001 INTO WA_T001.
    WRITE : /5 V_BOX AS CHECKBOX,
           WA_T001-BUKRS,
           WA_T001-BUTXT,
           WA_T001-ORT01,
           WA_T001-LAND1.
  CLEAR WA_T001.
ENDLOOP.
ENDIF.
```

* AT USER-COMMAND. *

AT USER-COMMAND.

CASE SY-UCOMM.

WHEN 'SALL'. "Select All

V_BOX = 'X'. "SELECT AND DISPLAY THE RECORDS

LOOP AT IT_T001 INTO WA_T001.

WRITE : /5 V_BOX AS CHECKBOX,

WA_T001-BUKRS,

WA_T001-BUTXT,

WA_T001-ORT01,

WA_T001-LAND1.

CLEAR WA_T001.

ENDLOOP.

WHEN 'DALL'. "DeSelect All

V_BOX = SPACE. "SELECT AND DISPLAY THE RECORDS

LOOP AT IT_T001 INTO WA_T001.

WRITE : /5 V_BOX AS CHECKBOX,

WA_T001-BUKRS,

WA_T001-BUTXT,

WA_T001-ORT01,

WA_T001-LAND1.

CLEAR WA_T001.

ENDLOOP.

WHEN 'DOWNLOAD'.

REFRESH IT_SELECTED_T001.

DESCRIBE LIST NUMBER OF LINES V_LINES.

DO V_LINES TIMES.

READ LINE SY-INDEX FIELD VALUE V_BOX

WA_T001-BUKRS

WA_T001-BUTXT

```
WA_T001-ORT01
WA_T001-LAND1
WA_T001-WAERS.

IF V_BOX = 'X'.
  APPEND WA_T001 TO IT_SELECTED_T001.
ENDIF.
ENDDO.

IF IT_SELECTED_T001 IS INITIAL.
  WRITE / 'NO RECORDs Selected to DownLoad'.
ELSE.

  CALL FUNCTION 'GUI_DOWNLOAD'
  EXPORTING
    FILENAME          = 'C:\DOWNLOAD.TXT'
    WRITE_FIELD_SEPARATOR = 'X'
  TABLES
    DATA_TAB        = IT_SELECTED_T001.
  IF SY-SUBRC <> 0.
    WRITE : / 'ITAB IS NOT SUCCESSFULLY DOWNLOAD'.
  ELSE.
    WRITE : / 'ITAB IS SUCCESSFULLY DOWNLOAD TO
C:\DOWNLOAD.TXT'.
  ENDF.

ENDIF.

WHEN 'CUST'.
  REFRESH IT_SELECTED_T001.

  DESCRIBE LIST NUMBER OF LINES V_LINES.

  DO V_LINES TIMES.
    READ LINE SY-INDEX FIELD VALUE V_BOX
      WA_T001-BUKRS
      WA_T001-BUTXT
      WA_T001-ORT01
      WA_T001-LAND1
```

Reports
We Never Compromise In Quality, Would You?

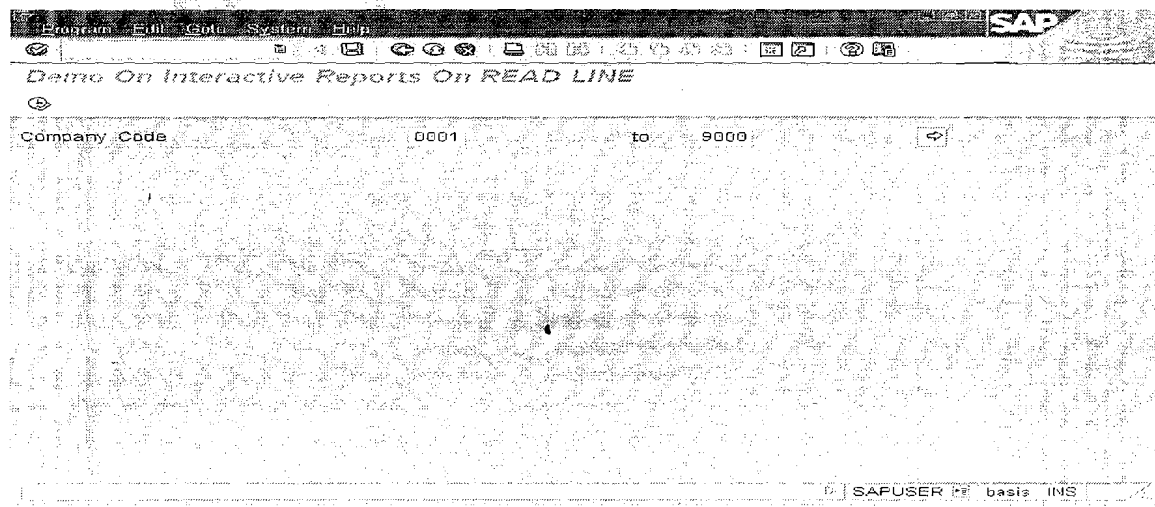
```
WA_T001-WAERS.
IF V_BOX = 'X'.
  APPEND WA_T001 TO IT_SELECTED_T001.
ENDIF.
ENDDO.

IF IT_SELECTED_T001 IS INITIAL.
  WRITE / 'NO Company Codes are Selected'.
ELSE.
  SELECT BUKRS KUNNR ZTERM INTO TABLE IT_KNB1
    FROM KNB1
    FOR ALL ENTRIES IN IT_SELECTED_T001
    WHERE BUKRS = IT_SELECTED_T001-BUKRS.
  IF IT_SELECTED_T001 IS INITIAL.
    WRITE / 'NO CUSTOMERS FOUND'.
  ELSE.
    LOOP AT IT_KNB1 INTO WA_KNB1.
      WRITE : / WA_KNB1-BUKRS,WA_KNB1-KUNNR,WA_KNB1-ZTERM.
    ENDLOOP.
  ENDIF.
ENDIF.

ENDCASE.
```

EXECUTE THE PROGRAM :

INPUT SCREEN



Reports

We Never Compromise In Quality, Would You?

System Help SAP

Demo to Work With AT USER COMMAND

Download Select All DeSelect All Display Customers

Demo to Work With AT USER COMMAND

<input type="checkbox"/>	0001	SAP A.S.	Waldorf	DE
<input type="checkbox"/>	1000	IDES AG	Frankfurt	DE
<input type="checkbox"/>	2000	IDES UK	London	GB
<input type="checkbox"/>	2100	IDES Portugal	Lisbon	PT
<input type="checkbox"/>	2200	IDES France	Paris	FR
<input type="checkbox"/>	2300	IDES España	Barcelona	ES
<input type="checkbox"/>	2400	IDES Italia	Milano	IT
<input type="checkbox"/>	2500	IDES Netherlands	Rotterdam	NL
<input type="checkbox"/>	3000	IDES US INC	New York	US
<input type="checkbox"/>	3010	Euro Subsidiary - Belgium	Brussels	BE
<input type="checkbox"/>	4000	IDES Canada	Toronto	CA
<input type="checkbox"/>	4500	Canadian Company	Toronto	CA
<input type="checkbox"/>	5000	IDES Japan	Tokyo	JP
<input type="checkbox"/>	6000	IDES México, S.A. de C.V.	México DF	MX
<input type="checkbox"/>	7000	IDES Brazil	São Paulo	BR
<input type="checkbox"/>	7500	IDES Argentina	Buenos Aires	AR
<input type="checkbox"/>	7600	IDES Columbia	Columbia	CO
<input type="checkbox"/>	7700	IDES Venezuela	Venezuela	VE
<input type="checkbox"/>	7800	IDES Peru	Perú	PE
<input type="checkbox"/>	8000	IDES Chile	Chile	CL

DEV (1) (800) ganapathu INS

A) Select the check box of the record(S) which have to be downloaded and click on the Export/Download Button.

System Help SAP

Demo to Work With AT USER COMMAND

Download Select All DeSelect All Display Customers

Demo to Work With AT USER COMMAND

Download to File (Shift+F2)

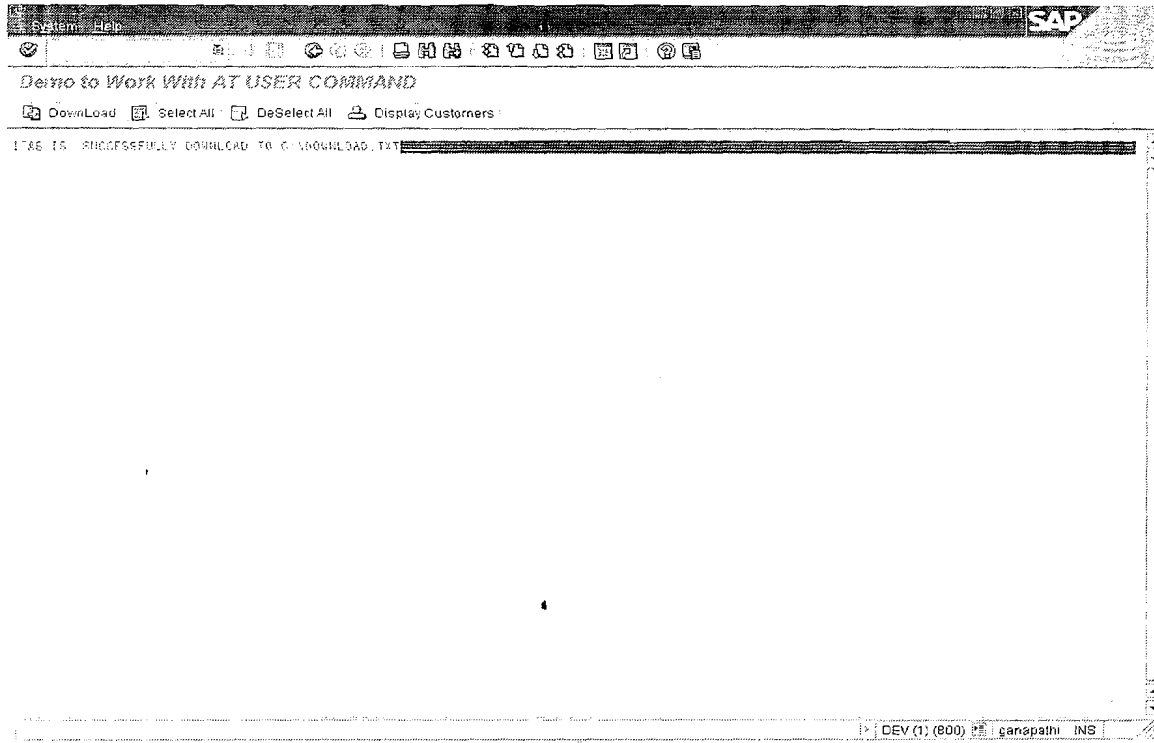
<input type="checkbox"/>	0001	SAP A.S.	Waldorf	DE
<input checked="" type="checkbox"/>	1000	IDES AG	Frankfurt	DE
<input checked="" type="checkbox"/>	2000	IDES UK	London	GB
<input type="checkbox"/>	2100	IDES Portugal	Lisbon	PT
<input type="checkbox"/>	2200	IDES France	Paris	FR
<input type="checkbox"/>	2300	IDES España	Barcelona	ES
<input type="checkbox"/>	2400	IDES Italia	Milano	IT
<input type="checkbox"/>	2500	IDES Netherlands	Rotterdam	NL
<input checked="" type="checkbox"/>	3000	IDES US INC	New York	US
<input type="checkbox"/>	3010	Euro Subsidiary - Belgium	Brussels	BE
<input type="checkbox"/>	4000	IDES Canada	Toronto	CA
<input type="checkbox"/>	4500	Canadian Company	Toronto	CA
<input checked="" type="checkbox"/>	5000	IDES Japan	Tokyo	JP
<input checked="" type="checkbox"/>	6000	IDES México, S.A. de C.V.	México DF	MX
<input type="checkbox"/>	7000	IDES Brazil	São Paulo	BR
<input type="checkbox"/>	7500	IDES Argentina	Buenos Aires	AR
<input type="checkbox"/>	7600	IDES Columbia	Columbia	CO
<input type="checkbox"/>	7700	IDES Venezuela	Venezuela	VE
<input type="checkbox"/>	7800	IDES Peru	Perú	PE
<input type="checkbox"/>	8000	IDES Chile	Chile	CL

DEV (1) (800) ganapathu INS

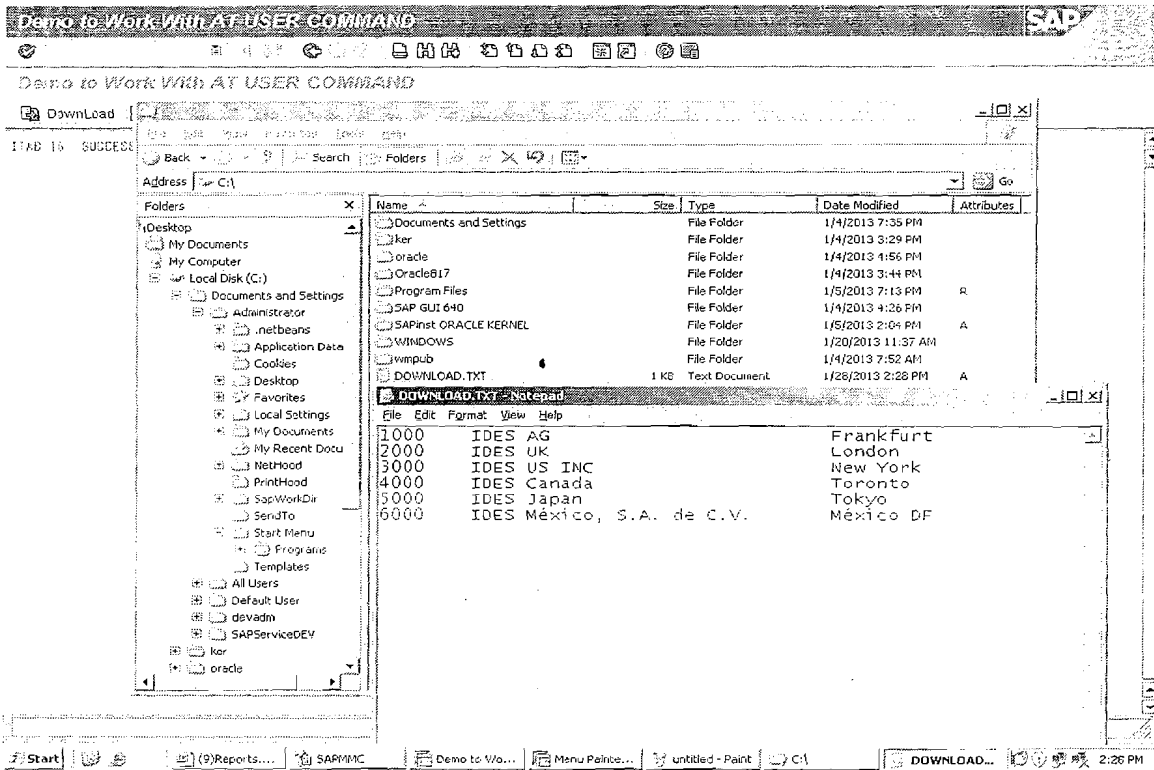
Start SAPMNC Demo to Work Wit... C:\Documents and Se... (9)Reports.doc - Micr... untitled - Paint 11:52 PM

Reports

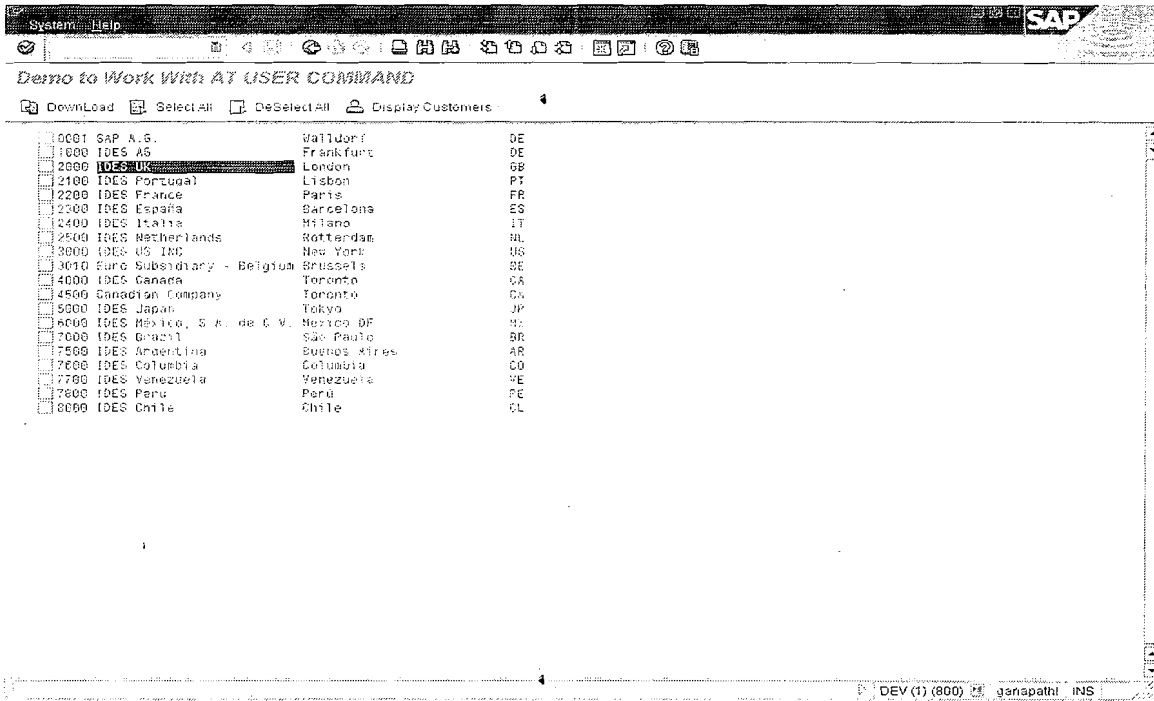
We Never Compromise In Quality, Would You?



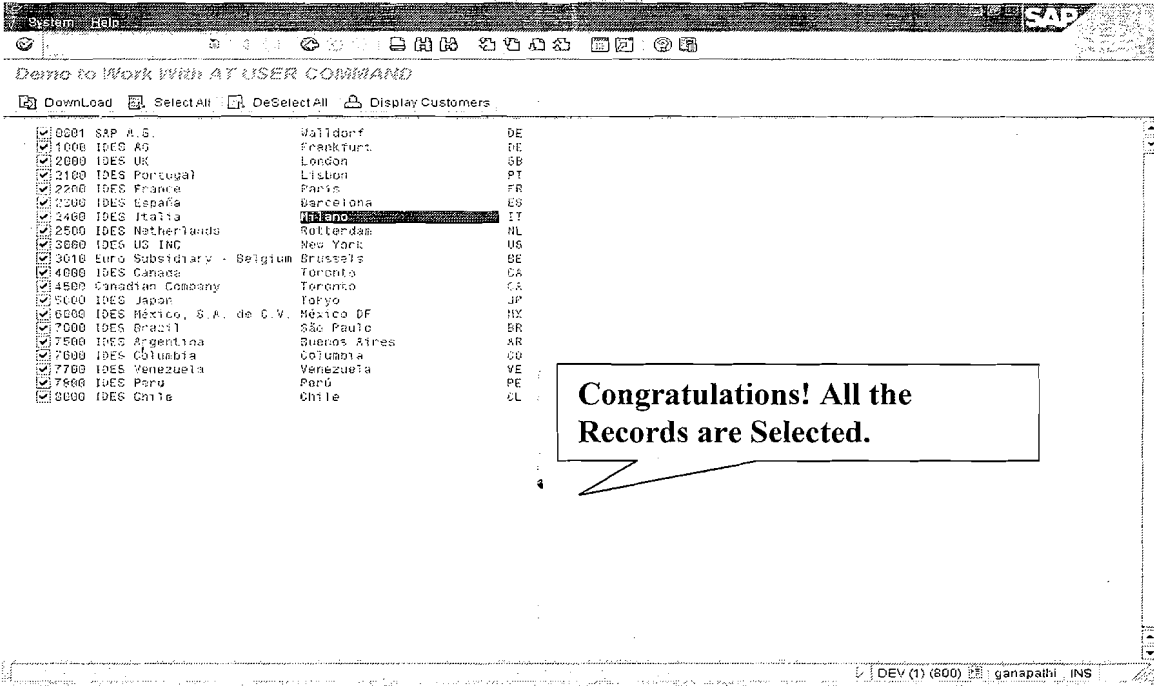
Check the File to See the records that are downloaded .



**B) CHECK THE FUNCTIONALITY OF SELECT ALL
EXECUTE THE PROGRAM**



Click On Select All



C) Check for the Functionality Of DALL(DeSelect All)  DeSelect All
From the Current Output Screen

Reports

We Never Compromise In Quality, Would You?

System Help SAP

Demo to Work With AT USER COMMAND

Download Select All DeSelectAll Display Customers

Company Code	Company Name	Country	Status
0001	SAP A.G.	Frankfurt	DE
1000	IDES AS	Frankfurt	DE
2000	IDES UK	London	GB
2100	IDES Portugal	Lisbon	PT
2200	IDES France	Paris	FR
2300	IDES España	Barcelona	ES
2400	IDES Italia	Milano	IT
2500	IDES Netherlands	Rotterdam	NL
3000	IDES US INC	New York	US
3010	Euro Subsidiary - Belgium	Brussels	BE
4000	IDES Canada	Toronto	CA
4500	Canadian Company	Toronto	CA
5000	IDES Japan	Tokyo	JP
6000	IDES México, S.A. de C.V.	México DF	MX
7000	IDES Brazil	São Paulo	BR
7500	IDES Argentina	Buenos Aires	AR
7600	IDES Columbia	Columbia	CO
7700	IDES Venezuela	Venezuela	VE
7800	IDES Peru	Perú	PE
8000	IDES Chile	Chile	CL

DEV (1) (800) ganapathi INS

Start SAPMMC Demo to Work With A... (9)Reports1.doc - Micros... untitled - Paint 11:57 PM

System Help SAP

Demo to Work With AT USER COMMAND

Download Select All DeSelectAll

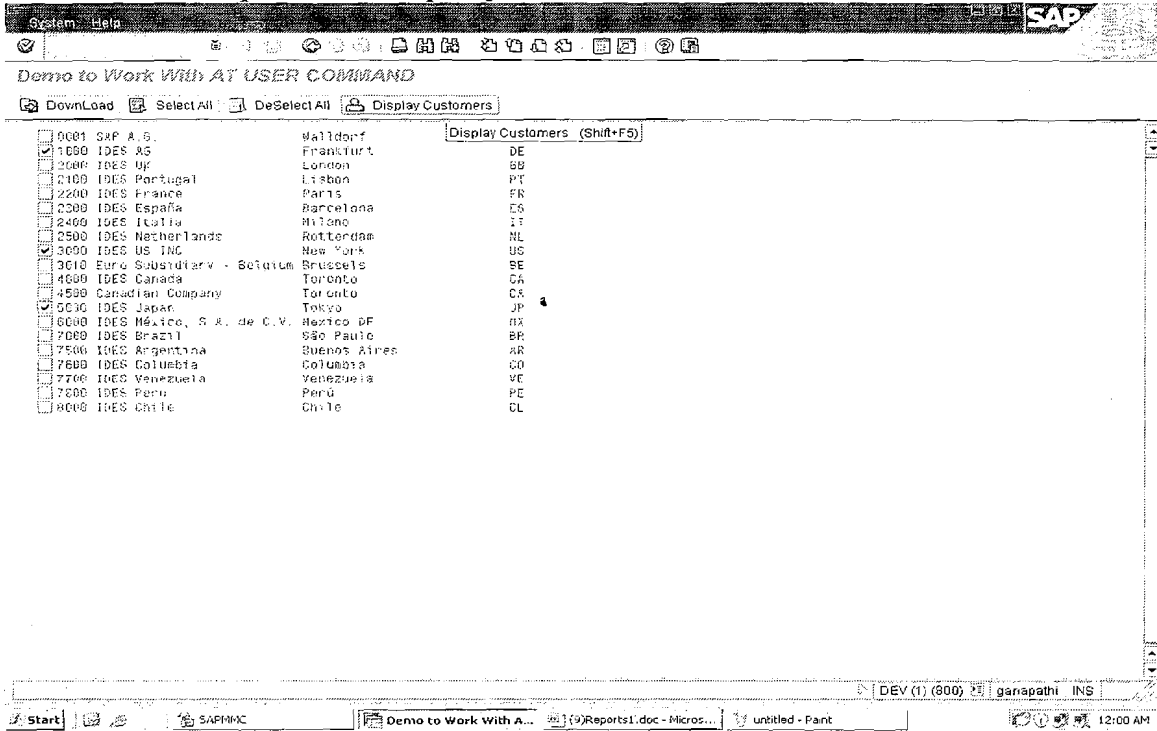
Company Code	Company Name	Country	Status
0001	SAP A.G.	Frankfurt	DE
1000	IDES AS	Frankfurt	DE
2000	IDES UK	London	GB
2100	IDES Portugal	Lisbon	PT
2200	IDES France	Paris	FR
2300	IDES España	Barcelona	ES
2400	IDES Italia	Milano	IT
2500	IDES Netherlands	Rotterdam	NL
3000	IDES US INC	New York	US
3010	Euro Subsidiary - Belgium	Brussels	BE
4000	IDES Canada	Toronto	CA
4500	Canadian Company	Toronto	CA
5000	IDES Japan	Tokyo	JP
6000	IDES México, S.A. de C.V.	México DF	MX
7000	IDES Brazil	São Paulo	BR
7500	IDES Argentina	Buenos Aires	AR
7600	IDES Columbia	Columbia	CO
7700	IDES Venezuela	Venezuela	VE
7800	IDES Peru	Perú	PE
8000	IDES Chile	Chile	CL
1001	IDES AG & Co. KG	Frankfurt	DE
C6FB	Training	SAPLand	DE
C6ED	CASO	SAPLand	DE
C6H1	FIBH Schweiz	Schweiz	CH
OPFO	Food Food	Chicago	US
F100	Bankhaus Frankfurt	Frankfurt	DE
F300	Liberty Bank	New York	US
R100	IDES Retail GmbH	Gießen	DE
R300	IDES Retail INC US	Los Angeles	US
S300	IDES Services	Atlanta	US

Notice that All the Records Are Deslected.

DEV (1) (800) ganapathi INS

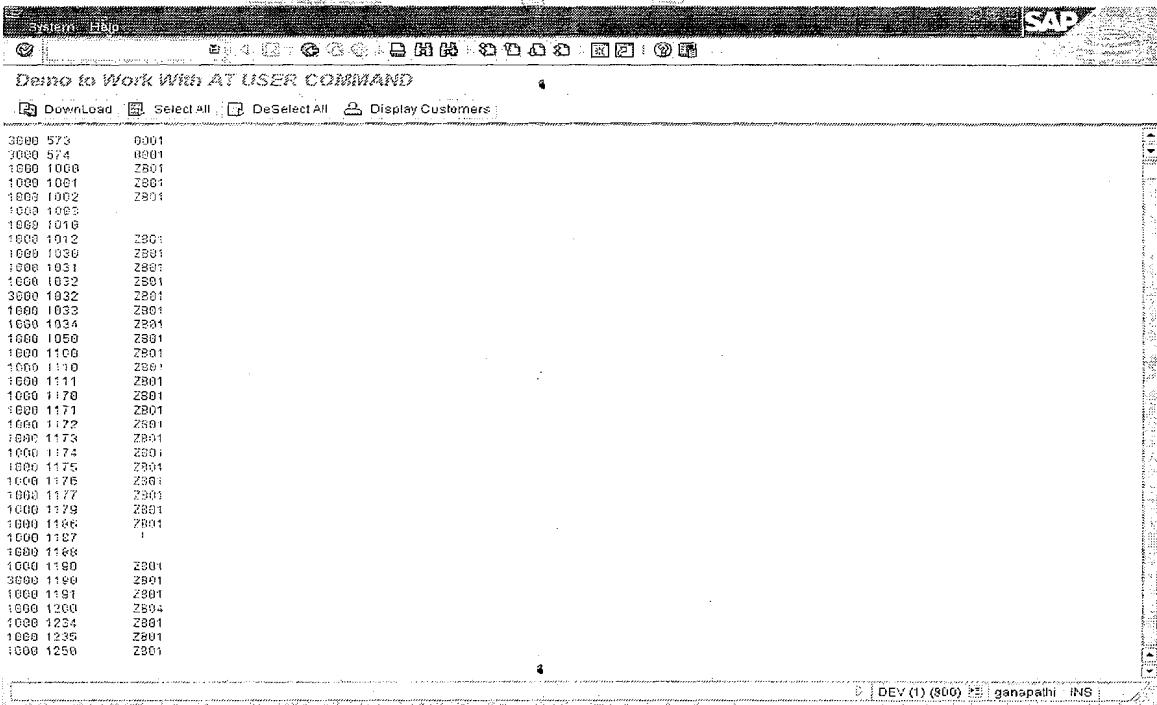
D) Display the List of Customers From all the Selected Company Codes.

Select the Required Company Codes and Click On  Display Customers



The screenshot shows the SAP menu bar with 'System Help' on the left and 'SAP' on the right. Below the menu bar, the title bar reads 'Demo to Work With AT USER COMMAND'. The main menu area contains several options: 'DownLoad', 'Select All', 'DeSelect All', and 'Display Customers'. The 'Display Customers' option is highlighted with a mouse cursor. A tooltip for 'Display Customers (Shift+F5)' is visible over the option.

List Of Customers For the Selected Company Codes



The screenshot shows the SAP menu bar with 'System Help' on the left and 'SAP' on the right. Below the menu bar, the title bar reads 'Demo to Work With AT USER COMMAND'. The main menu area contains several options: 'DownLoad', 'Select All', 'DeSelect All', and 'Display Customers'. The 'Display Customers' option is highlighted with a mouse cursor. A tooltip for 'Display Customers (Shift+F5)' is visible over the option.

0001	SAP A.S.	Waldorf	DE
1000	IDES AS	Frankfurt	DE
2000	IDES UK	London	GB
2100	IDES Portugal	Lisbon	PT
2200	IDES France	Paris	FR
2300	IDES España	Barcelona	ES
2400	IDES Italia	Milano	IT
2500	IDES Netherlands	Rotterdam	NL
3000	IDES US INC	New York	US
3010	Euro Subsidiary - Belgium	Brussels	BE
4000	IDES Canada	Toronto	CA
4500	Canadian Company	Toronto	CA
5000	IDES Japan	Tokyo	JP
6000	IDES México, S. A. de C.V.	Mexico DF	MX
7000	IDES Brazil	São Paulo	BR
7500	IDES Argentina	Buenos Aires	AR
7600	IDES Colombia	Colombia	CO
7700	IDES Venezuela	Venezuela	VE
7800	IDES Peru	Peru	PE
8000	IDES Chile	Chile	CL

Exercises

- I. Display the list of Monthly Sales orders and the Total Value of the Sales for each Sales Organization and Under each Company Code and the Grand Total. (Use Control Break Statements).
Input : Select-Options for Sales Organization,
 Select-Options for Date
(Default Dates : Begin and End Dates of Current month).

- II. Display the list of Monthly Purchase orders and the Total Value of the Purchase orders for each Purchasing Organization and Under each Company Code and the Grand Total . (Use Control Break Statements).
Input : Select-Options for Purchasing Organization,
 Select-Options for Date
(Default Dates : Begin and End Dates of Current month).

- III. Create an interactive report for displaying Vendor information. Based on the selection made , the corresponding vendor bank detail are displayed such that the line selected in the basic list was visible along with secondary list.

- IV. Create an report to display a list of purchase requisitions with details like MRP controller, release date and unit of measure along with standard details.

- V. Create a report displaying Customer number, Name, Material No., Quantity, Description on selection 'Goods Receipt No.'

- VI. Create interactive report, to list all the sales that took place during the month for particular material.

- VII. Create a interactive list for purchase requisitions at a given plant.

- VIII. Create an interactive report to display list Company Codes(Basic List), Customers under the Selected Company Code(1st 2nd ry), Customer sales orders for particular customer, items for particular order.

- IX. Create a report which lists delivery number, delivery quantity, customer number, material number and material description for a given shipping plant.

Reports
We Never Compromise In Quality, Would You?

- X. Create a report that shows a list of purchase requisition and purchase orders for a selected vendor, by material group listing by material.**
- XI. Create a report to get list of purchase orders created only On Saturdays and Sundays during the particular period.**
- XII. Create an interactive report that list out all the materials for a given plant. Secondary list contains vendor details who supplies the chosen material.**
- XIII. Create a report to get list of Sales orders created only On Saturdays and Sundays during the particular period.**



10. Modularization Techniques

Duration in Days - 2(* 2 Hrs)

a. Introduction

b. INCLUDE Programs

c. MACROs

d. Subroutines

e. Function Modules

Modularization : Is to Sub divide the Main Program into Reusable Modules or Blocks.

Note : Modularization Improves the Re-usability(Reusing ability) and Readability(Reading ability) of the Source Code.

Example :

Program : without Modules

```
Statement A.  
Statement B.  
Statement C.  
Statement D.  
Statement E.  
Statement B.  
Statement C.  
Statement D.  
Statement F.  
Statement B.  
Statement C.  
Statement D.  
Statement G.
```

With Modularization

```
Statement A.  
Call Module M.  
Statement E.  
Call Module M.  
Statement F.  
Call Module M.  
Statement G.
```

```
MODULE M.  
Statement B.  
Statement C.  
Statement D.  
ENDMODULE.
```

Note : Instead Of Repeating Statement B.
Statement C.
Statement D.

It is Better to Group them as a Re-usable Module and Call it any no Of Times.

Modularization Techniques :

- A) Those that can be called by ABAP statements in ABAP programs.
- B) Called from outside a program by the ABAP runtime system.

A) Processing blocks that are called from ABAP programs in Detail:

- a. Function modules
- b. Methods(Discussed in Object Oriented ABAP)
- c. Subroutines
- d. Macros
- e. INCLUDE Programs

Working With Include Programs:

Include programs are global R/3 Repository objects. They are solely for modularizing source code, and have no parameter interface.

Creating Our Own Include Programs :

If you create an include program yourself, you must assign it the Program type I in its program attributes. You can also create or change an include program by double-clicking on the name of the program after the INCLUDE statement in your ABAP program. **If the program exists, the ABAP Workbench navigates to it. If it does not exist, the system creates it for you.**

An include program cannot run independently, but Can be Included into other programs. Include programs can contain other includes.

Note : For the syntax check to produce valid results, you must check the program in which the include occurs.

Using Include Programs

To use an include program in another program, enter the statement

INCLUDE <Name Of the include Program>.

The INCLUDE statement has the same effect as copying the source code of the include program <incl> into the program. The INCLUDE statement must be the only statement on a line and cannot extend over several lines.

EXAMPLE PROGRAM ON INCLUDES

```
*&-----*
*& Report  ZGDEMO_INCLUDES                               *
*&                                               *
*&-----*
*&                                               *
*&                                               *
*&-----*

REPORT  ZGDEMO_INCLUDES

INCLUDE ZGDEMO_TOP.

V_RESULT = P_INPUT1 + P_INPUT2.

WRITE : / 'THE RESULT OF ADDITION IS', V_RESULT.
```

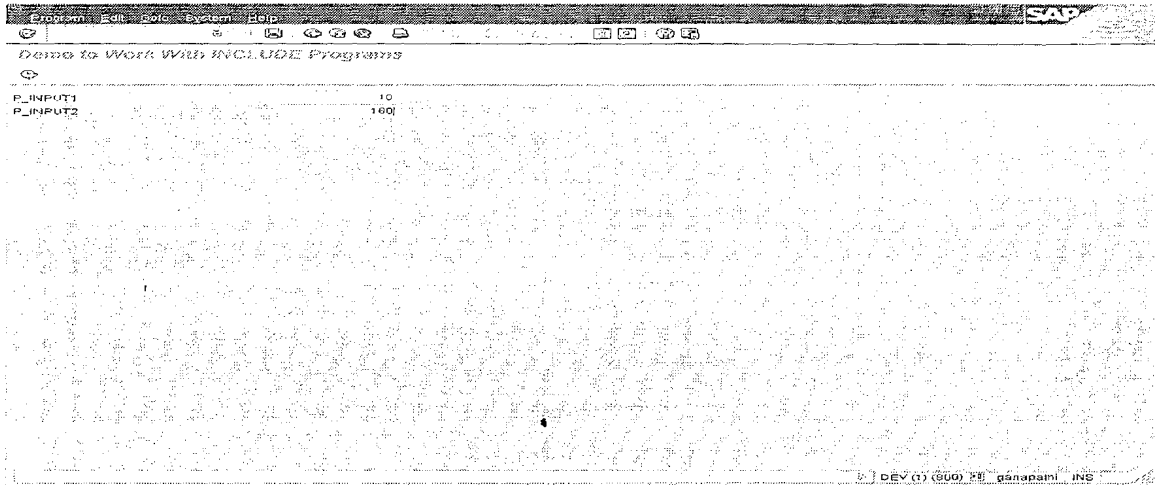
INCLUDE PROGRAM DETILS :

```
*&-----*  
*& Include          ZGDEMO_TOP          *  
*&-----*
```

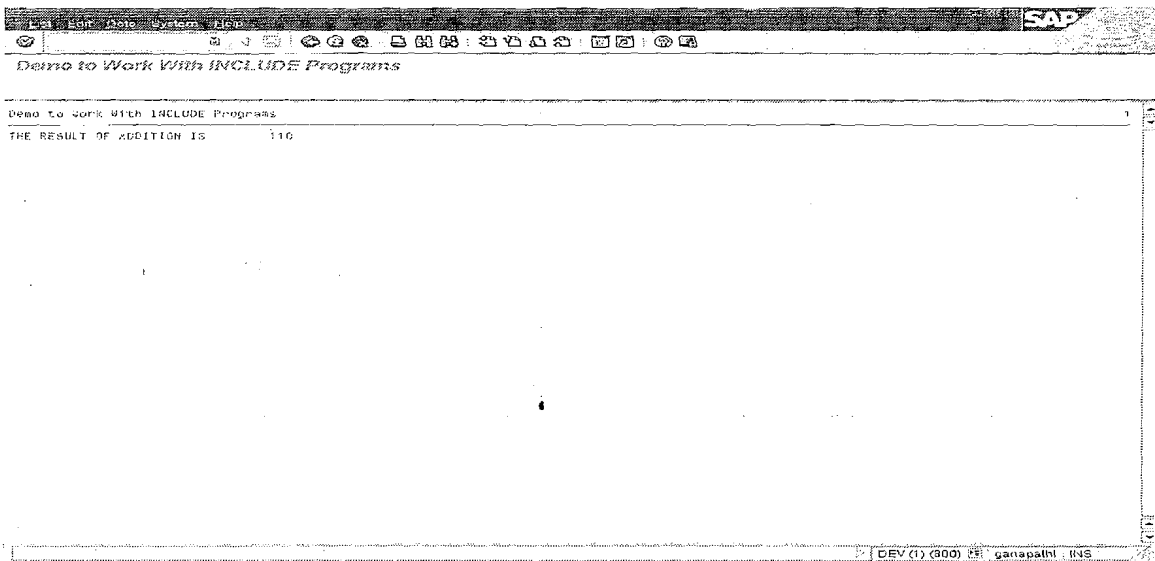
PARAMETER : P_INPUT1 TYPE I,
P_INPUT2 TYPE I.

DATA V_RESULT TYPE I.

EXECUTE THE MAIN PROGRAM :



EXECUTE



Working with Function Modules:

Function Modules & Features :

Function modules are for **global modularization**, that is, they are always called from a different program. Function modules contain functions that are used by many different programs.

- 1) Each FM Should be linked to one function groups (special ABAP programs with type F) , Which Acts as a MAIN Program For all the FMs which are stored in the Same Function Group.
- 2) Transaction Code to Create Function Groups and Modules is **SE37**.
- 3) Are Global reusable Components i.e they Can be called from any Program without including the Definition Details.
- 4) Function modules allow to encapsulate and reuse global **functions** in the R/3 System.
- 5) **FMs** are stored in a **Central Library**. The R/3 System contains a wide range of predefined function modules that can be called from any ABAP program.
- 6) **The Definition Of FMs Can be tested as an Individual Component Before Calling them in any Program**
- 7) **FM Definition Can be Debugged before Calling**
- 8) **Function modules** also support exception handling. This allows you to catch certain errors while the function module is running.

Function Groups in Detail:

Notes:

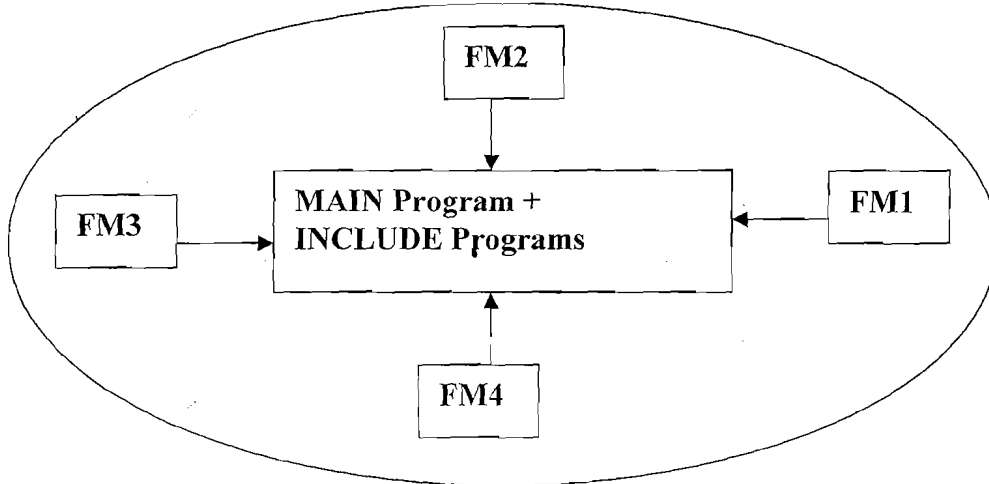
Function groups are containers for function modules. You cannot execute a function group. When you call a function module, the system loads the whole of its function Modules into the internal session of the calling program So that it is Better to group Only the related Function Modules. When the function group is Created, The main program and include programs are generated automatically.

The main program **SAPL<fgrp>** contains nothing but the **INCLUDE** statements for the following include programs:

- **L<fgrp>TOP**. This contains the **FUNCTION-POOL** statement (equivalent for a function group of the **REPORT** or **PROGRAM** statement) and global data declarations for the entire function group.

- **L<fgrp>UXX**. This contains further INCLUDE statements for the include programs **L<fgrp>U01, L<fgrp>U02, ...** These includes contain the actual function modules.
- The include programs **L<fgrp>F01, L<fgrp>F02, ...** can contain the coding of
- subroutines that can be called from all function modules of the group.

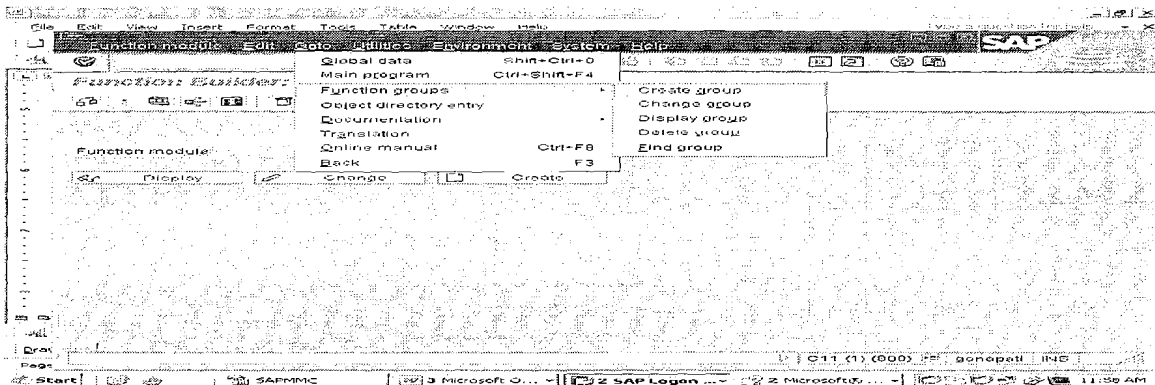
Function Group



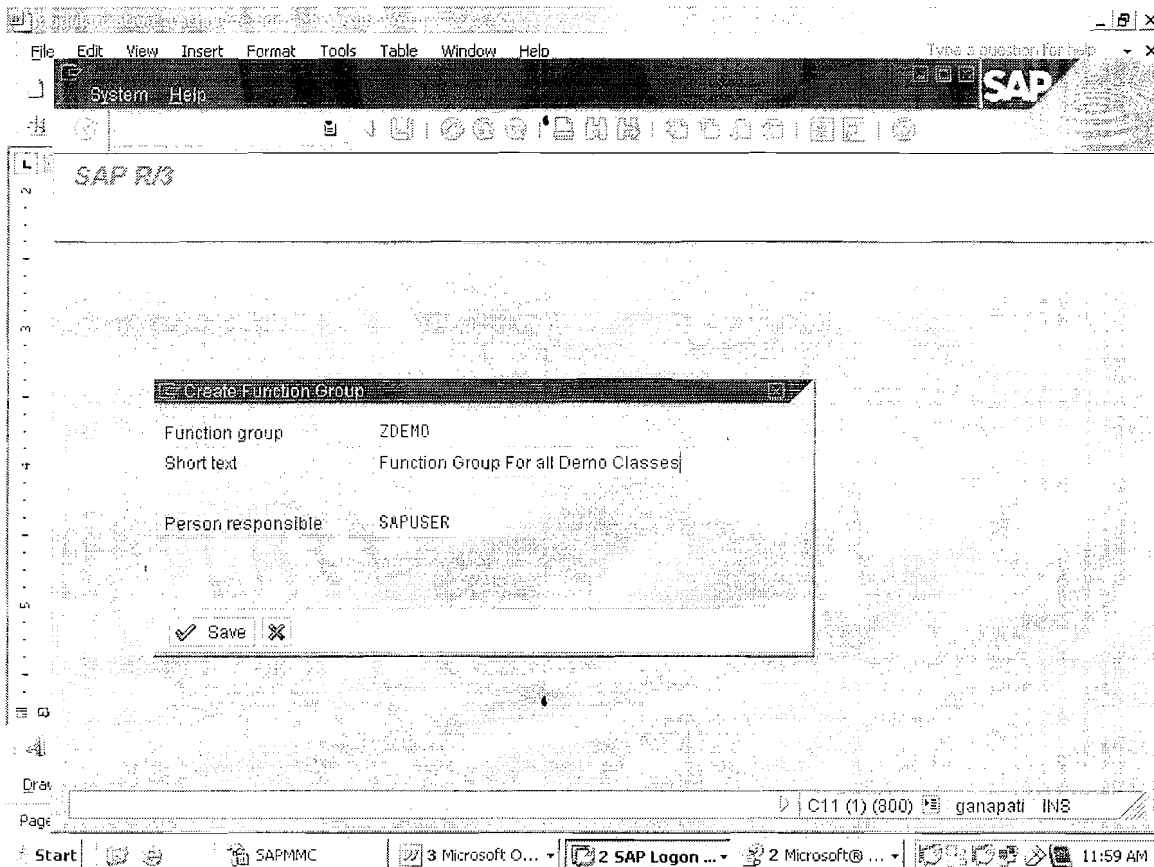
Note: All of (Only) the function modules in a function group can access the **global data** of the group. For this reason, **you should place all function modules that use the same data in a single function group**. Example, if you have a set of function modules that all use the same internal table, you could place them in a function group containing the table definition in its global data.

Steps to Work with the Function Group:

- Execute SE37
- Choose *Goto -> Function groups -> Create group*.



1. Specify the function group name and a short text.



2. Choose *Save*.
3. Activate the Function Group.

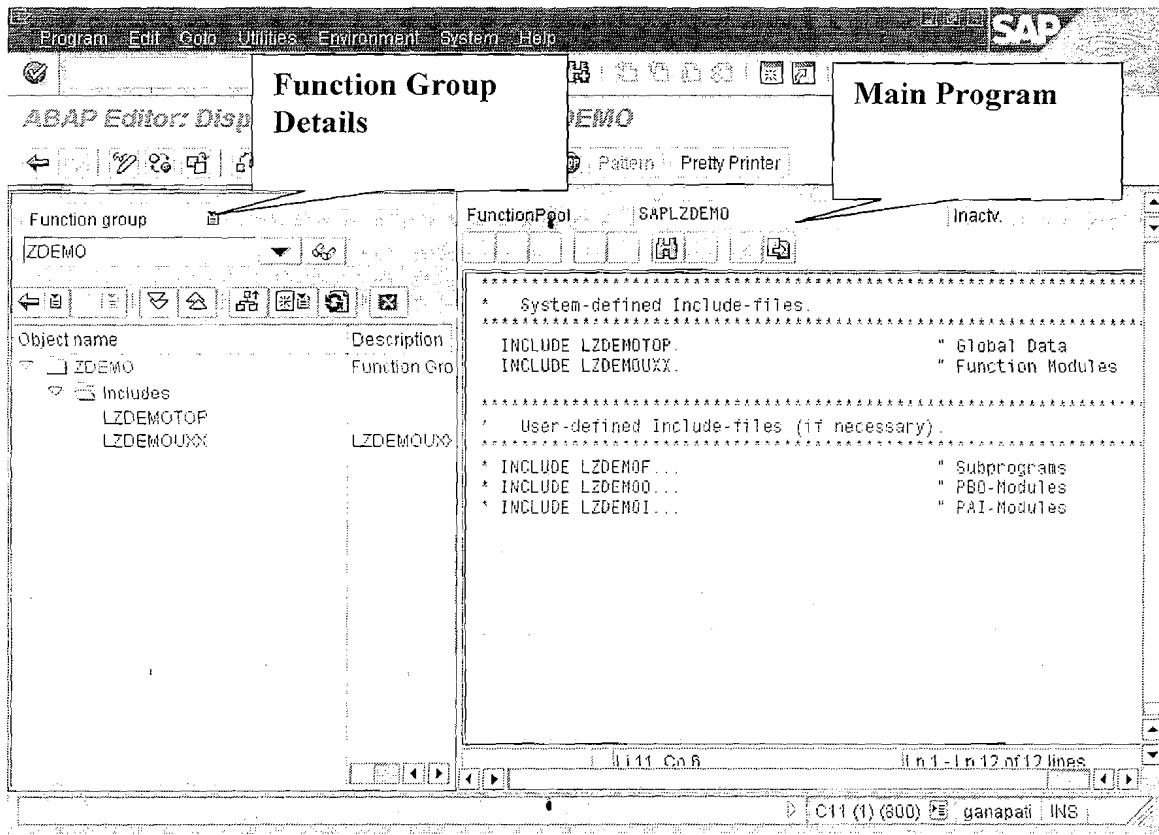
Note: Except in 4.7EE, Function Group Activation is Not Default. Instead it has to be activated explicitly using the below Procedure.

To Activate it , Open the Function Group in **SE80** so that we can check all the MAIN and INCLUDE Programs and **then Activate it**.

Execute SE80 and Enter the Function Group Name ZDEMO and it Displays all the Programs Included .

Modularization Techniques

We Never Compromise in Quality, Would You?



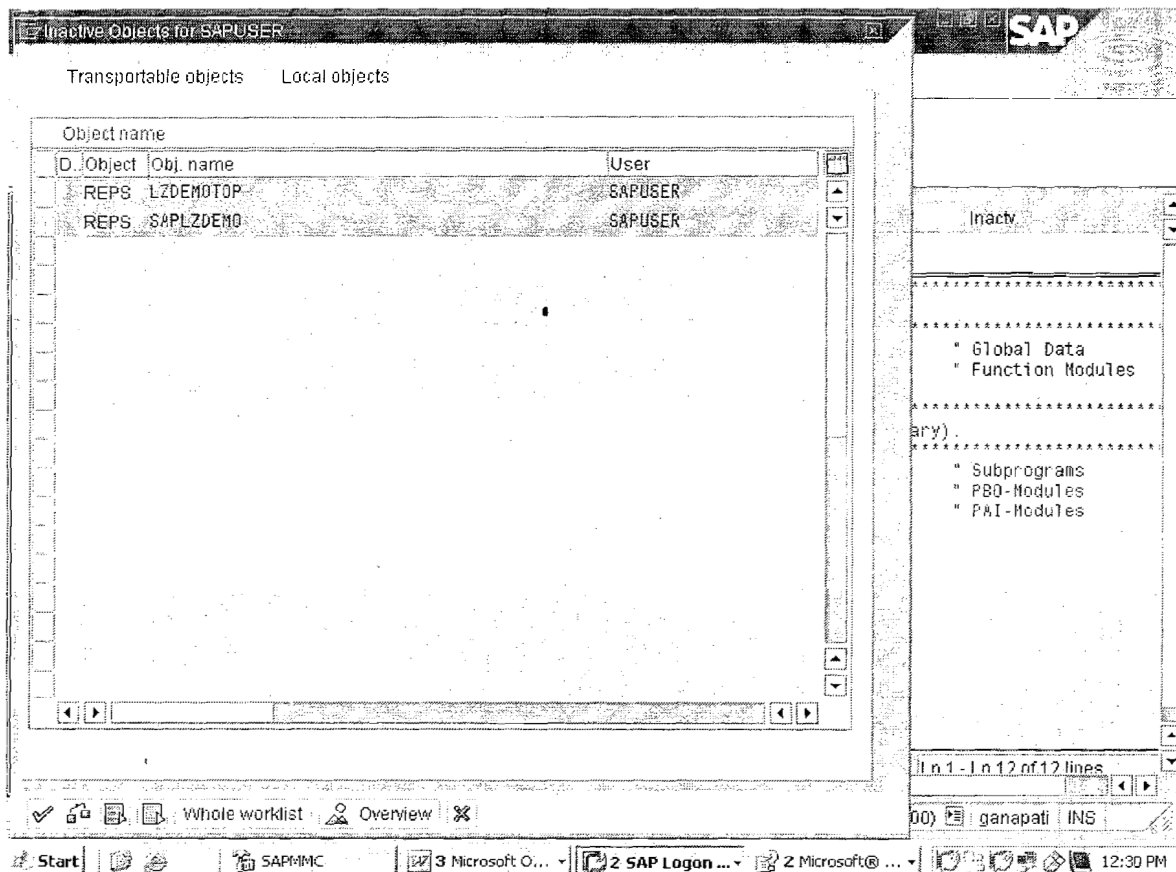
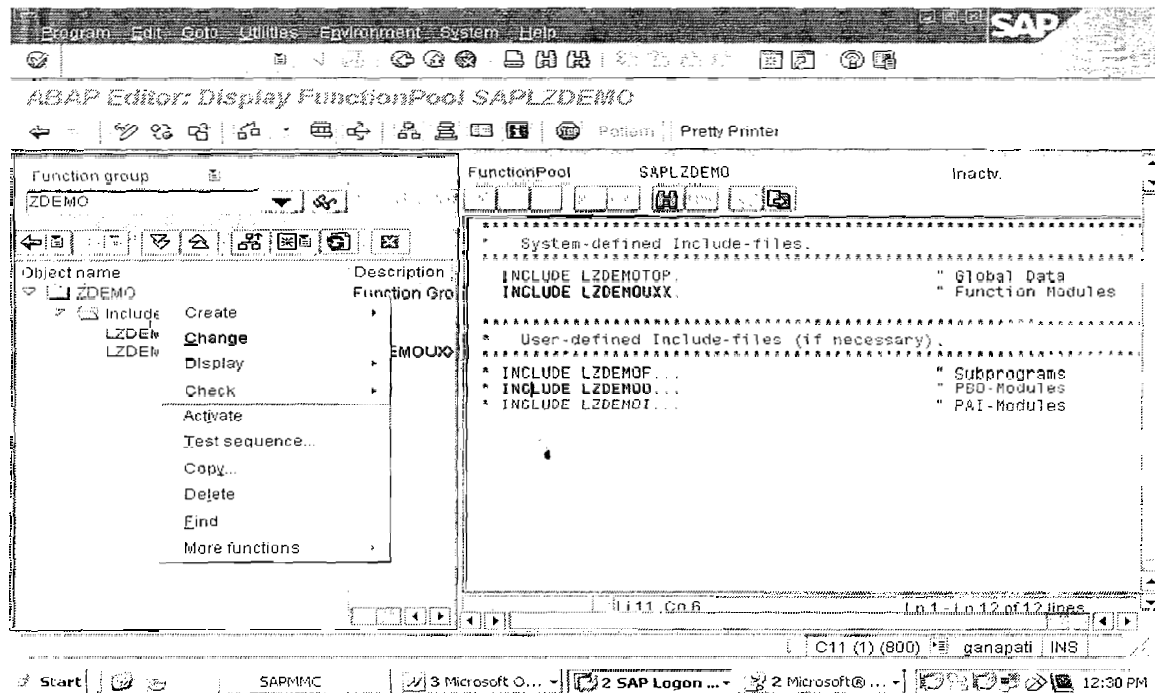
The name of the main program is assigned by the system. This is made up of the prefix **SAPL** followed by the function group name. For example, the main program for function group ZDEMO is called SAPLZDEMO.

The names of the include files begin with **L** followed by the name of the function group, and conclude with **UXX** (or **TOP** for the **TOP** include). The TOP include contains *global data* declarations that are used by all of the function modules in the function group. The other include file within the main program is used to hold the function modules within the group.

To **Activate the Function Group**, Select the Function Group, Right Click, Activate it.

Modularization Techniques

We Never Compromise in Quality, Would You?



Function Module in Detail :

Function Module Attributes:

- **Documentation**

The documentation describes the **purpose of the function module**, lists the parameters for passing data to and from the module, and the exceptions. It tells you how you can pass data to and from the function module, and which errors it handles.

- **Interface parameters and exceptions**

Import(INPUT) parameters. These must be supplied with data when you call the function module, unless they are flagged as optional. **You cannot change them in the function module.**

- **Export(OUTPUT) parameters.** These pass data from the function module back to the calling program. Export parameters **are always optional**. You do not have to receive them in your program.
- **Changing parameters.** These must be supplied with data when you call the function module, unless they are flagged as optional. They can be changed in the function module. The changed values are then returned to the calling program.

Note : Changing Parameters Acts as both INPUT & OUTPUT.

- **Tables parameters.** You use these to pass internal tables. They are treated like CHANGING parameters. However, you can also pass internal tables with other parameters if you specify the parameter type appropriately.

Note 1 : You can specify the types of the interface parameters, either by referring to **ABAP Dictionary types or elementary ABAP types**. **When you call a function module, you must ensure that the actual parameter and the interface parameters are compatible.**

Note 2 : **Interface parameters are, by default, passed by value.** However, they can also be passed by reference. Tables parameters can only be passed by reference. You can assign default values to optional importing and changing parameters. If an optional parameter is not passed in a function module call, it either has an initial value, or is set to the default value.

Note 3 : Exceptions are used to handle errors that occur in function modules. The calling program checks whether any errors have occurred and then takes action accordingly.

Creating Function Modules

You can only create function modules using the Function Builder (SE37).

Steps to Work with the Function Modules:

1. Check whether a suitable function module already exists. If not, proceed to step 2.
2. Create a function group, if no appropriate group exists yet.
3. Create the function module.
4. Define the function module interface by entering its parameters and exceptions.
5. Write the actual ABAP code for the function module, adding any relevant global data to the TOP include.
6. Activate the module.
7. Test the module.
8. Document the module and its parameters for other users.
9. Release the module for general use.

Steps to Create Function Module In Detail :

Requirement:

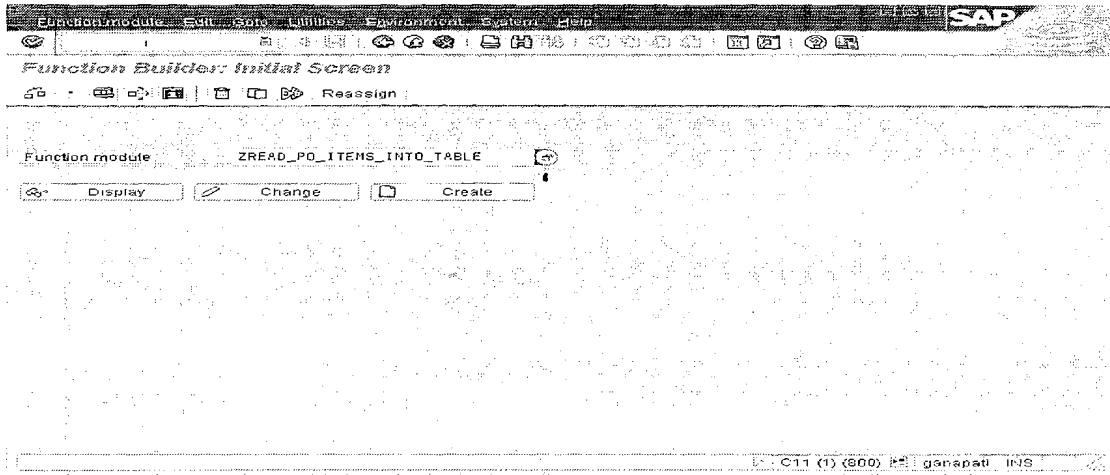
Create a Function Module to read the Line Items for the given Purchase Order.

Input : Purchase Order No

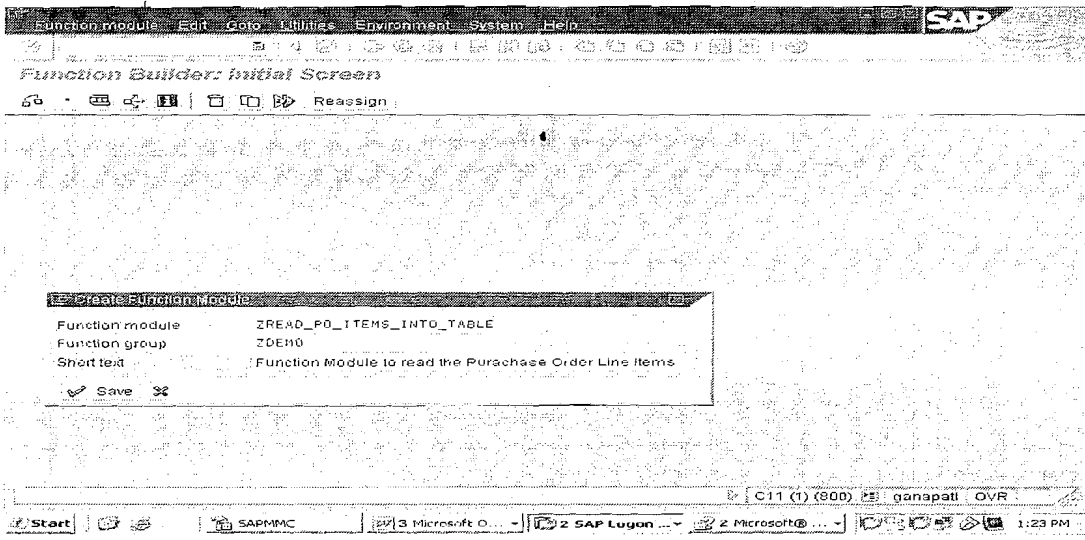
Output : An Internal table with all the Line Item Details(Item No, Material No, Quantity and Net Price) for the given Purchase Order.

Create a function module ZREAD_PO_ITEMS_INTO_TABLE to read Purchase Order Line Items Data for the Given Purchase Order Number from table EKPO into an internal table, which is then passed back to the calling program.

- 1) Execute SE37.
- 2) Enter the Name For Function Module: ZREAD_PO_ITEMS_INTO_TABLE
- 3) Choose Create.



Enter the Function Group Name, which we already Created and short text.



Choose SAVE.

Parameter Interface :

Since function modules can be used anywhere in the system, their interfaces can only contain references to data types that are declared system wide. These are the elementary ABAP data types, the systemwide generic types, such as ANY TABLE, and types defined in the ABAP Dictionary. **You cannot use LIKE to refer to data types declared in the main program.**

Click On **IMPORT** and provide the input for Purchase Doc.No

Function module: ZREAD_PO_ITEMS_INTQ_TABLE Inactive (revised)

Attributes Import Export Changing Tables Exceptions Source code

Parameter Name	Type	Associated Type	Default value	Opt.	Pa.	Short text
IM_EBELN	TYPE	EBELN		<input type="checkbox"/>	<input type="checkbox"/>	Purchasing Document Number
				<input type="checkbox"/>	<input type="checkbox"/>	
				<input type="checkbox"/>	<input type="checkbox"/>	
				<input type="checkbox"/>	<input type="checkbox"/>	
				<input type="checkbox"/>	<input type="checkbox"/>	
				<input type="checkbox"/>	<input type="checkbox"/>	

To pass data back to the calling program, **the function module needs an EXPORT parameter as an internal table type**. For this, we define a system wide **Table type ZIT_EKPO** with the line type **ZEKPO** in the ABAP Dictionary.

Note: Internal Tables Can be passed through **TABLES** Also. But it is not recommended In Object Oriented Scenarios.

Ex: IT_ITEMS LIKE EKPO. (Using TABLES)

Click On **EXPORT** Parameter and Provide the Output Internal Table.

Function Builder: Change ZREAD_PO_ITEMS_INTQ_TABLE

Function module: ZREAD_PO_ITEMS_INTQ_TABLE

Attributes Import Export Changing

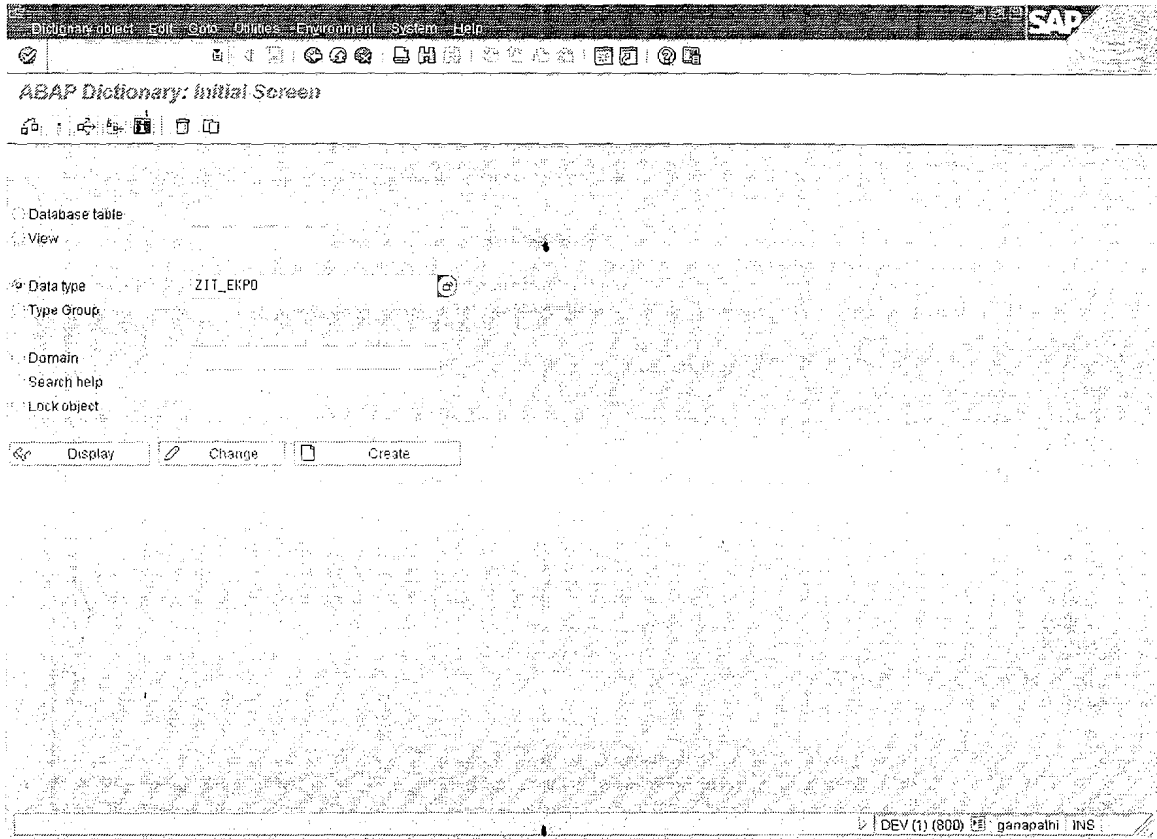
ZIT_EKPO Should be created as a Table Type In DDIC.

Parameter name	Type spec.	Reference type	Default value	Short text	Long text
IT_ITEMS	TYPE	ZIT_EKPO		Table for Purchase Order Line Items	Org.

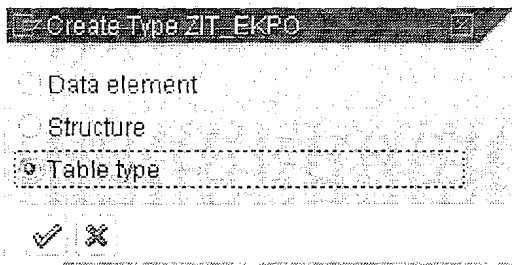
Steps to Create Table Type in DDIC:

EXECUTE SE11

ENTER NAME FOR DATATYPE



CREATE.



CONTINUE WITH Table Type Option

Modularization Techniques

We Never Compromise in Quality, Would You?

The screenshot shows the SAP 'Table Type' configuration interface. The title bar includes 'Table Type', 'Edit', 'Goto', 'Utilities', 'Environment', 'System', and 'Help'. The main window title is 'Dictionary: Maintain Table Type'. Below the title bar, there are navigation icons and a 'Hierarchy Display' button. The main content area is divided into several sections:

- Table Type:** ZIT_EKPO, Status: New(Revised)
- Short text:** TABLE TYPE FOR EKPOI
- Attributes:** Line Type, Initialization and Access, Key
- Line Type:** EKPOI
- Built-in type:** Data Type, No. of Characters: 0, Decimal Places: 0
- Reference type:** Reference to Predefined Type, Data Type, Length: 0, Decimal Places: 0

At the bottom of the window, there is a status bar with the following information: 'The object was created in the original language English (EN)', 'DEV (1) (300)', and 'ganapathi INS'.

SAVE, CHECK AND ACTIVATE IT.


```
*" EXPORTING
*" REFERENCE(IT_ITEMS) TYPE ZIT_EKPO
*" EXCEPTIONS
*" NOT_FOUND
*"-----
-----
```

```
SELECT * INTO TABLE IT_ITEMS
        FROM EKPO
WHERE EBELN = IM_EBELN.
```

```
IF SY-SUBRC <> 0.
MESSAGE E007(ZDEMO) RAISING NOT_FOUND.
ENDIF.
```

```
ENDFUNCTION.
```

Raising Exceptions

There are two ABAP statements for raising exceptions. **They can only be used in function modules:**

RAISE <except>. and

MESSAGE..... RAISING <except>.

The effect of these statements depends on whether the calling program handles the exception or not. If the name <except> of the exception or OTHERS occurs in the EXCEPTIONS addition of the CALL FUNCTION statement, the exception is handled by the calling program.

If the calling program does not handle the exception

- The RAISE statement terminates the program and switches to debugging mode.
- The MESSAGE RAISING statement display the specified message. How the processing continues depends on the message type.

If the calling program handles the exception, both statements return control to the program. No values are transferred. The MESSAGE RAISING statement does not display a message. Instead, it fills the system fields SY-MSGID, SY-MSGTY, SY-MSGNO, and SY-MSGV1 to SY-MSGV4.

Data in Function Modules


You can use the TYPES and DATA statements to create local data types and objects. The interface parameters also behave like local data objects. In addition, you can access all of

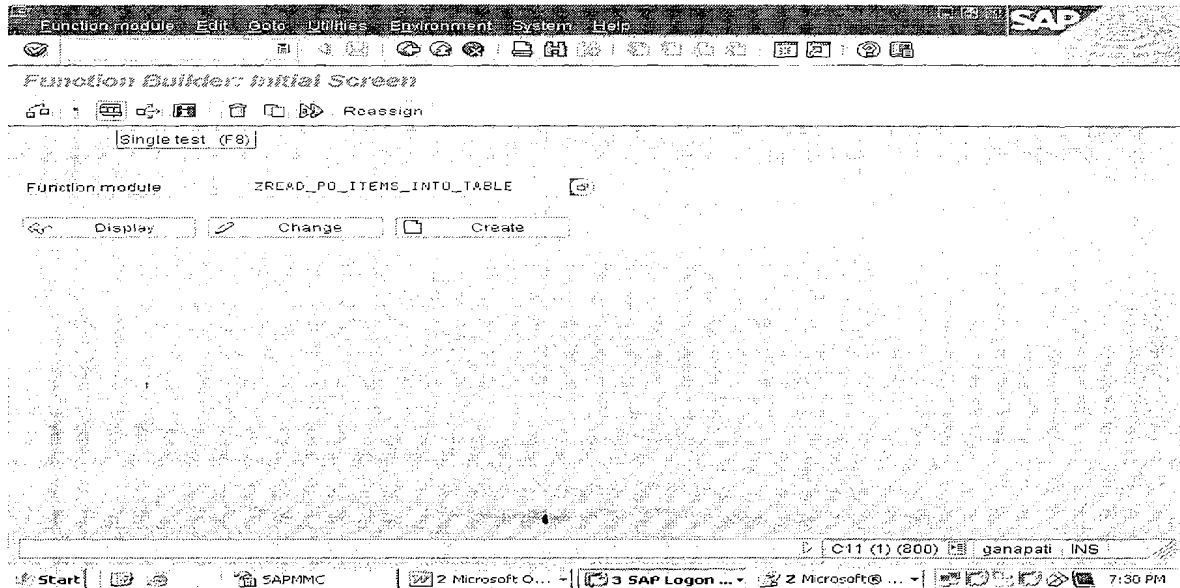
the global data of the main program. This data is defined in the include program L<fgrp>TOP. To open this include, choose *Goto -> Global data*.

Testing Function Modules

We should use the test environment in the Function Builder to test new function modules before releasing them for general use.

Choose *Test* from the Function Builder initial screen.

Execute SE37 and Provide the Function Module name and click On  (F8).

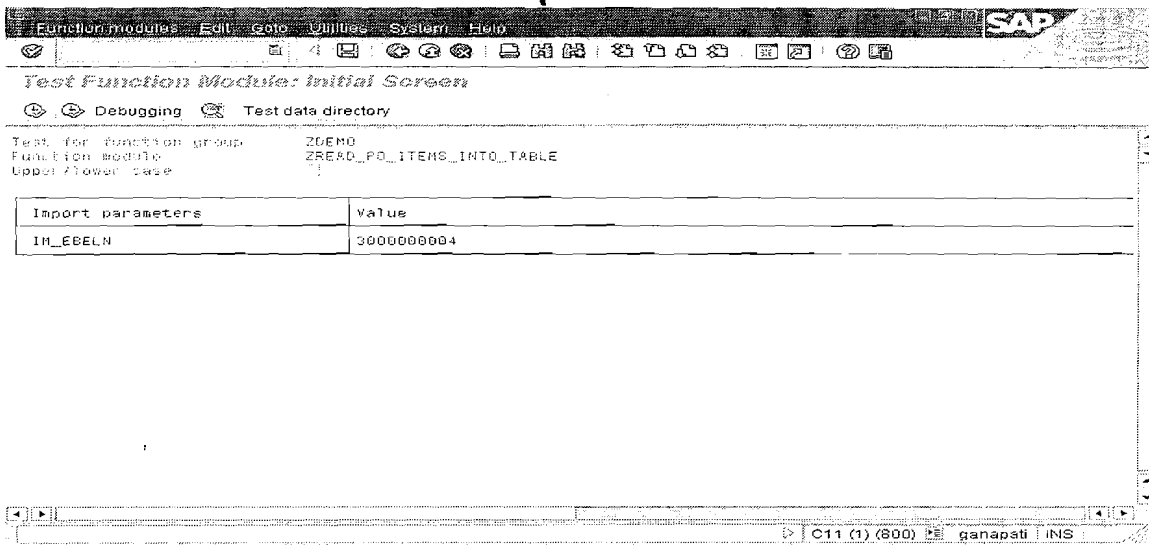


PROVIDE THE INPUT

NOTE: Fill Values for the relevant import, changing, and tables parameters. To fill in single-field parameters, enter the value in the displayed field. To fill in table/ structure parameters, double-click on the parameter name.

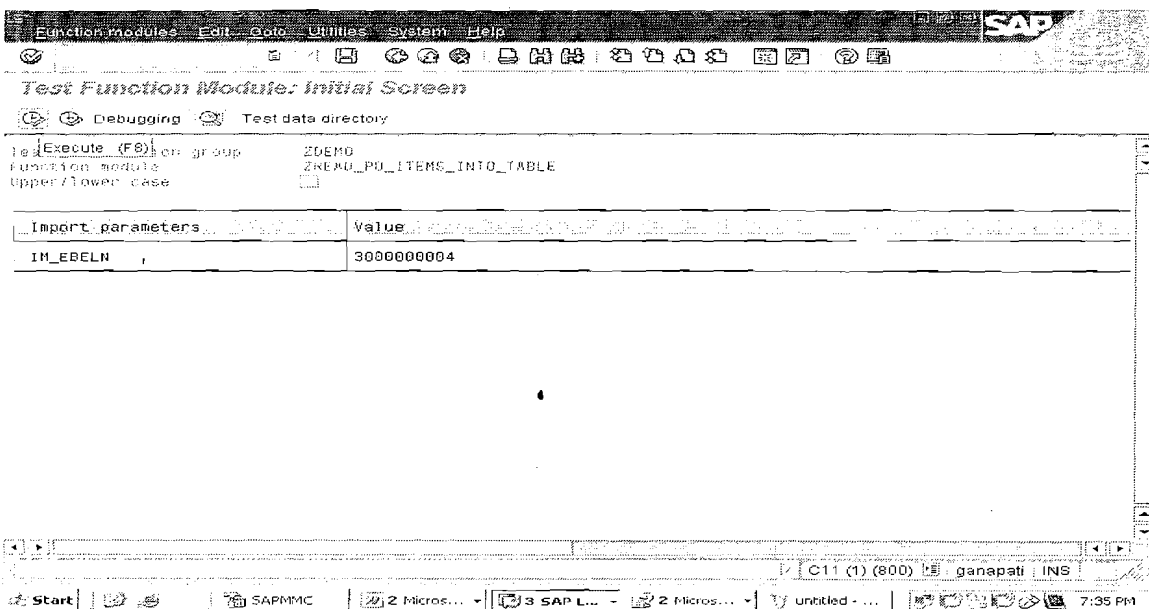
Modularization Techniques

We Never Compromise in Quality, Would You?



Choose *Execute*.

The system runs the function module using your input and displays the values of the export parameters that result:



Execution Results :

Modularization Techniques

We Never Compromise in Quality, Would You?

Function modules Edit Goto Utilities System Help

Test Function Module: Result Screen

Test for function group ZDEMO
Function module ZREAD_PO_ITEMS_INTO_TABLE
Upper/lower case

Runtime: 738 Microseconds

Import parameters	Value
IN_EBELN	3000000004

Export parameters	Value
IT_ITEMS	2 Entries

Start | SAPMMCM | 2 Micros... | 3 SAP L... | 2 Micros... | untitled - ... | 7:35 PM

Function modules Edit Goto Utilities System Help

Test Function Module: Result Screen

Test for function group ZDEMO
Function module ZREAD_PO_ITEMS_INTO_TABLE
Upper/lower case

Runtime: 738 Microseconds

Import parameters	Value
IN_EBELN	3000000004

Export parameters	Value
IT_ITEMS	2 Entries

Click On this, to check the Exporting Parameters Contents

Start | SAPMMCM | 2 Micros... | 3 SAP L... | 2 Micros... | untitled - ... | 7:35 PM

Modularization Techniques

We Never Compromise in Quality, Would You?

Structure Editor: Display IT_ITEMS from Entry 1

MAM	EBELN	EBELP	L	S	REDAT	TXZ01	MATR	EMATH	BUKR	WERK	LGOR	BEDNR
800	3800000004	00001			02.11.2000	Desk Pads			3000	3200		
800	3000000004	00002			02.11.2000	Mouse Pads			3000	3200		

DEV (1) (800) ganapati INS

Test Case2 :

Execute the Function Module for the Given Input.

Test Function Module: Initial Screen

Debugging Test data directory

Test for function group ZDEMO
function module ZREAD_PO_ITEMS_INTO_TABLE
Upper/lower case

Import parameters	Value
IN_EBELN	5000000014

C11 (1) (800) ganapati INS

Execute it.

Function modules Edit Goto Utilities System Help **SAP**

Test Function Module: Result Screen

Test for function group: ZDEMO
Function module: ZREAD_PO_ITEMS_INTO_TABLE
Upper/lower case:

Runtime: 627 Microseconds

Exception: NOT_FOUND
Message ID: ZDEMO
Message: No Records Found For the Given Selection

Message: Since No Records found for the given PO Number, it raises the exception

Import parameters	Value
IM_EBELN	5000000014

Export parameters	Value
IT_ITEMS	<input type="checkbox"/> 0 Entries

C11 (1) (800) ganapati INS

Calling Function Modules From ABAP Programs

You can call a function module from within any ABAP program by using the following ABAP statement:

CALL FUNCTION <function module>

EXPORTING

f₁ = a₁

...

f_n = a_n

IMPORTING

f₁ = a₁

....

f_n = a_n

[CHANGING f₁ = a₁.... f_n = a_n]

[TABLES f₁ = a₁.... f_n = a_n]

[EXCEPTIONS e₁ = r₁.... e_n = r_n

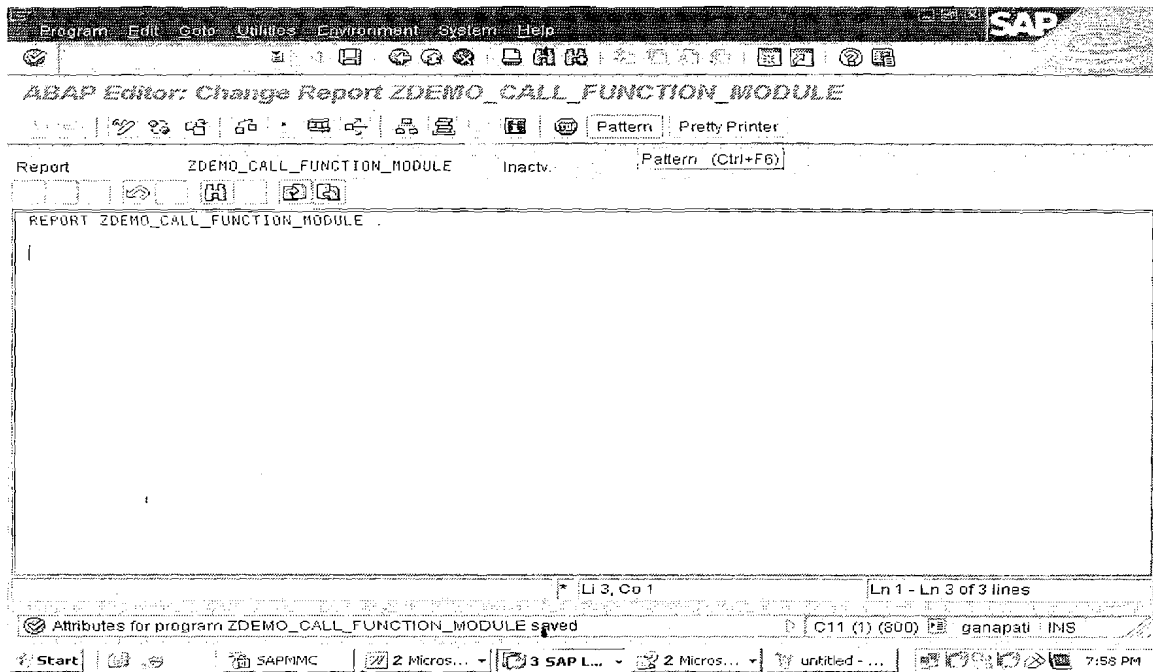
[ERROR_MESSAGE = r_E]

[OTHERS = r_o]

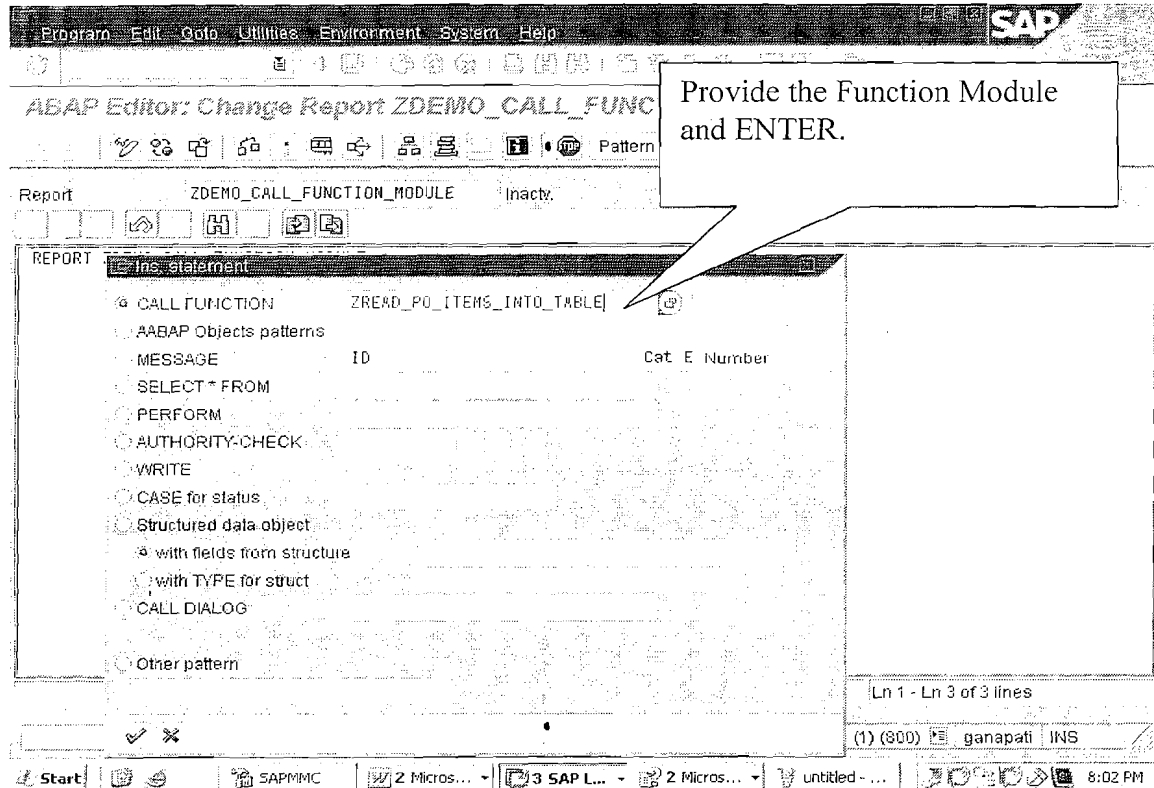
Note : You can use the function *Pattern* in the ABAP editor to call a function from within your coding.

Position the cursor at the point in your coding where you want to call the function.

1. Choose *Pattern*.



2. In the dialog box that appears, **mark** the selection field in front of CALL FUNCTION .
3. Enter the name of the function module in the input field.



4. Choose *Continue*.

The system inserts the function module with the interface into your coding.

5. Maintain the appropriate parameters and handle the exceptions.

Details about Interface Parameters:

- The **EXPORTING** option passes the actual parameter a_i to the formal input parameter f_i . **The formal parameters must be declared as import parameters in the function module.**
- The **IMPORTING** option passes the formal output parameter f_i of the function module to the actual parameter a_i . **The formal parameters must be declared as export parameters in the function module.** The parameters may have any data type.
- The **CHANGING** option passes the actual parameter a_i to the formal parameter f_i . After the function module has been processed, the system returns the (changed) values of the formal parameters f_i to the actual parameters f_i . The formal parameters must be declared as **CHANGING** parameters in the function module. These parameters, too, may have any data type.

- The TABLES option passes internal tables between the actual and formal parameters. **The internal tables are always passed by reference.** The parameters in this option must always reference internal tables.
- The EXCEPTIONS option allows you to react to errors in the function module. Exceptions are provided as special parameters in order to be able to react to possible error events during processing of the function module. When an exception occurs, function module processing terminates. Example: If exception e_i is triggered, the system stops processing the function module and does not pass any values back to the program. The calling program receives the exception e_i by assigning the value r_i as a return code to the system field **SY-SUBRC**.
- You can then evaluate the system field in the calling program.

```
Ex : CASE SY-SUBRC .  
      WHEN 0.  
          "SuccessFul  
      WHEN 1.  
          "Exception 1  
      WHEN 2.  
          "Exception 2.  
      .  
      WHEN OTHERS.  
          "Others  
      ENDCASE.
```

ABAP Program to Call the Function Module:

```
REPORT ZDEMO_CALL_FUNCTION_MODULE .
```

```
PARAMETER : P_EBELN TYPE EBELN.  
DATA      : IT_EKPO TYPE ZIT_EKPO,  
           WA_EKPO TYPE ZEKPO.
```

```
CALL FUNCTION 'ZREAD_PO_ITEMS_INTO_TABLE'  
EXPORTING  
  IM_EBELN    = P_EBELN  
IMPORTING  
  IT_ITEMS    = IT_EKPO  
EXCEPTIONS  
  NOT_FOUND   = 1  
  OTHERS      = 2
```

```
CASE sy-subrc .  
  WHEN 0.  
    LOOP AT IT_EKPO INTO WA_EKPO.  
      WRITE : / WA_EKPO-EBELN, "DOC NO
```

```
WA_EKPO-EBELP, "ITEM NO  
WA_EKPO-MATNR, "MATERIAL NO  
WA_EKPO-NETPR. "QUANTITY  
CLEAR WA_EKPO.  
ENDLOOP.
```

WHEN 1.

```
WRITE / 'NO DATA FOUND'.
```

ENDCASE.

OUTPUT :

Execute the above Program **ZDEMO_CALL_FUNCTION_MODULE**.

Working With Subroutines:

DAY-2

- Subroutines are principally for local modularization, that is, they are generally **called from the program in which they are defined**. You can use subroutines to write functions that are used **repeatedly within a program**. You can define subroutines in any ABAP program.

Note : Subroutines are normally called internally, that is, they contain sections of code or algorithms that are used frequently **locally**. **If you want a function to be reusable throughout the system, use a function module.**

Note: We Should know **TWO** things to work with any Modulrization Technique. **How to Define the Module and How to Call it.**

Note : Subroutines Should be Called **First** and Defined next in the Program.

No Statements Can be Executed ater

Calling the Subroutines:

```
PERFORM <Name> USING <INPUT1>
                        <INPUT2> etc
                        CHANGING <OUTPUT1>
                        <OUTPUT2> etc
```

All Arguments Passed at the time of Calling the Subroutines are Called **Actual Parameters.**

Defining Subroutines

```
FORM <Name> USING <FP_INPUT1>
                        <FP_INPUT2> etc
                        CHANGING <FP_OUTPUT1>
                        <FP_OUTPUT2> etc
```

All Arguments Passed at the time of Defining the Subroutines are Called **Formal Parameters.**

ENDFORM.

<Name> is the name of the subroutine. The optional additions USING is to Pass Inputs and CHANGING is to define OUTPUT Parameters. **subroutines cannot be nested. You should therefore place your subroutine definitions at the end of the program.**

The Parameter Interface

Note: We can pass data between Calling Program and Subroutines by Using Parameters.

The USING and CHANGING additions in the **FORM** statement define the **formal parameters** of a subroutine.

The USING and CHANGING additions in the **PERFORM** statement define the **Actual parameters** of a subroutine.

Within a subroutine, The value of the Formal parameters is the value passed from the corresponding actual parameter.

Subroutines can have the following formal parameters:

Parameters Passed by Reference (Call By Reference)

You list these parameters after USING or CHANGING without the VALUE addition:

FORM <subr> USING ... <pj>

CHANGING ... <pj> the formal parameter **occupies no memory of its own. During a subroutine call, only the address of the actual parameter is transferred to the formal parameter. i.e Simply Formal Parameter is another name given to the Actual Parameter. So that the Changes to the Formal Parameter reflects to the corresponding Actual Parameters.**

For calling by reference, USING and CHANGING are equivalent. For documentation purposes, you should use USING for INPUT Parameters which are not changed in the subroutine, and CHANGING for OUTPUT Parameters which are changed in the subroutine.

NOTE : To avoid the value of an actual parameter being changed automatically, you must pass it by value.

```
***** *
* PROGRAM : ZDEMO_SUBROUTINES_CALL_BY_REF .
* AUTHOR  : GANAPATI . ADIMULAM
* PURPOSE : WRITING SUBROUTINE USING CALL BY REFERENCE
***** *
```

REPORT ZDEMO_SUBROUTINES_CALL_BY_REF .

DATA V_COUNTER TYPE I.

PERFORM increment_counter changing v_counter.

WRITE : / 'THE COUNTER AFTER INCREMENT IS', V_COUNTER.

PERFORM increment_counter changing v_counter.

WRITE : / 'THE COUNTER AFTER INCREMENT IS', V_COUNTER.

PERFORM increment_counter changing v_counter.

WRITE : / 'THE COUNTER AFTER INCREMENT IS', V_COUNTER.

&-----

*& Form increment_counter

&-----

* text

* -->P_V_COUNTER text

form increment_counter changing fp_v_counter TYPE I.

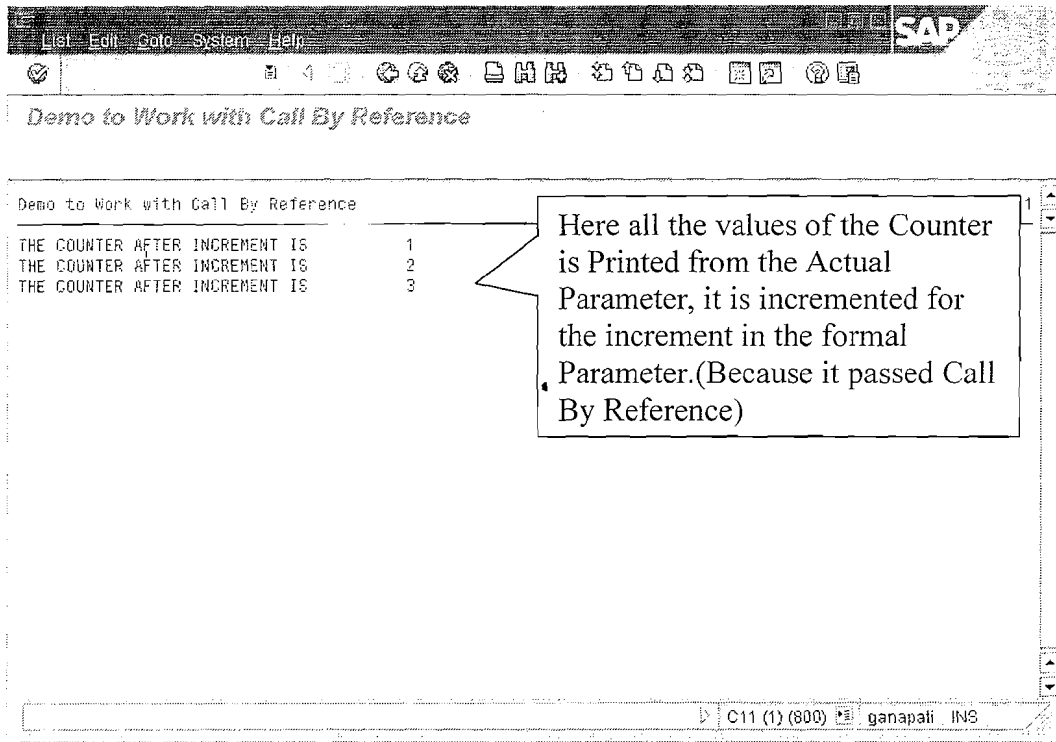
fp_v_counter = fp_v_counter + 1.

*NOTE : HERE THE CHANGES TO THE FORMAL PARAMETER REFLECTS

*TO THE ACTUAL PARAMETER V_COUNTER

endform. " increment_counter'

OUTPUT :



Input Parameters That Pass Values (Call by Value)

You list these parameters after USING with the VALUE addition:

```
FORM <subr> USING ... VALUE(<pi>)
```

The formal parameter occupies its own memory space. When you call the subroutine, the value of the actual parameter is passed to the formal parameter. If the value of the formal parameter changes, this has no effect on the actual parameter.

```
*****
* PROGRAM : ZDEMO_SUBROUTINES_CALL_BY_VAL
* AUTHOR  : GANAPATI . ADIMULAM
* PURPOSE : WRITING SUBROUTINE USING CALL BY VALUE
*****
REPORT ZDEMO_SUBROUTINES_CALL_BY_VAL .
DATA V_COUNTER TYPE I.

PERFORM increment_counter using v_counter.
WRITE : / 'THE COUNTER AFTER INGREMENT IS', V_COUNTER.

PERFORM increment_counter using v_counter.
WRITE : / 'THE COUNTER AFTER INCREMENT IS', V_COUNTER.
```

PERFORM increment_counter using v_counter.
 WRITE : / 'THE COUNTER AFTER INCREMENT IS', V_COUNTER.

```
*&-----*
*&   Form increment_counter
*&-----*
*   text
*-----*
*   -->P_V_COUNTER text
*-----*
```

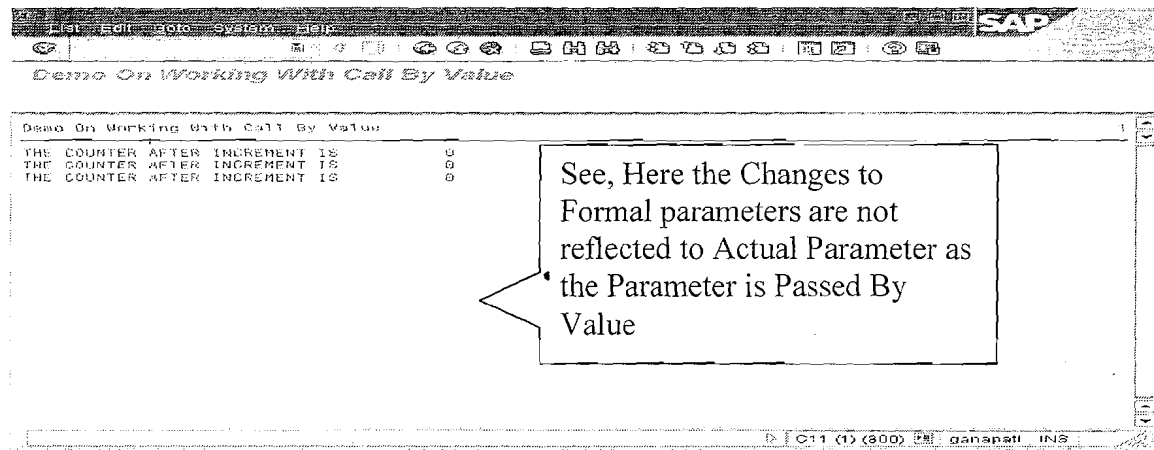
form increment_counter using VALUE(fp_v_counter) TYPE I.

fp_v_counter = fp_v_counter + 1.

*NOTE : HERE THE CHANGES TO THE FORMAL PARAMETER DOESN'T REFLECTS

*TO THE ACTUAL PARAMETER V_COUNTER AS IT IS CALL BY CAL endform. " increment_counter

OUTPUT :



Output Parameters That Pass Values(Call By Value & Return)

You list these parameters after CHANGING with the VALUE addition:

FORM <subr> CHANGING ... VALUE(<p_i>).

The formal parameter occupies its own memory space. When you call the subroutine, the value of the actual parameter is passed to the formal parameter. If the subroutine concludes successfully, that is, when the ENDFORM statement occurs, or when the subroutine is terminated through a CHECK or EXIT statement, the current value of the formal parameter is copied into the actual parameter.

If the subroutine terminates prematurely due to an error message, no value is passed. It only makes sense to terminate a subroutine through an error message in the PAI processing of a screen, that is, in a PAI module, in the AT SELECTION-SCREEN event, or after an interactive list event.

```
***** *
* PROGRAM : ZDEMO_SUBROUTINES_CALL_BY_VAL
* AUTHOR  : GANAPATI . ADIMULAM
* PURPOSE : WRITING SUBROUTINE USING CALL BY VALUE &
*         RETURN, I.E THE VALUE OF THE FORMAL PARAMETER
*         WILL BE TRANSFERED TO ACTUAL PARAMETER ONLY
*         AFTER THE IT COMES OUT THE SUBROUTINE
*         SUCCESSFULLY
***** *
```

```
REPORT ZDEMO_SUBROUTINES_CALL_BY_VAL .
DATA V_COUNTER TYPE I.
```

```
PERFORM increment_counter CHANGING v_counter.
WRITE : / 'THE COUNTER AFTER INCREMENT IS', V_COUNTER.
```

```
PERFORM increment_counter CHANGING v_counter.
WRITE : / 'THE COUNTER AFTER INCREMENT IS', V_COUNTER.
```

```
PERFORM increment_counter CHANGING v_counter.
WRITE : / 'THE COUNTER AFTER INCREMENT IS', V_COUNTER.
```

```
*&-----*
*&   Form increment_counter
*&-----*
*   text
*-----*
*   -->P_V_COUNTER text
*-----*
```

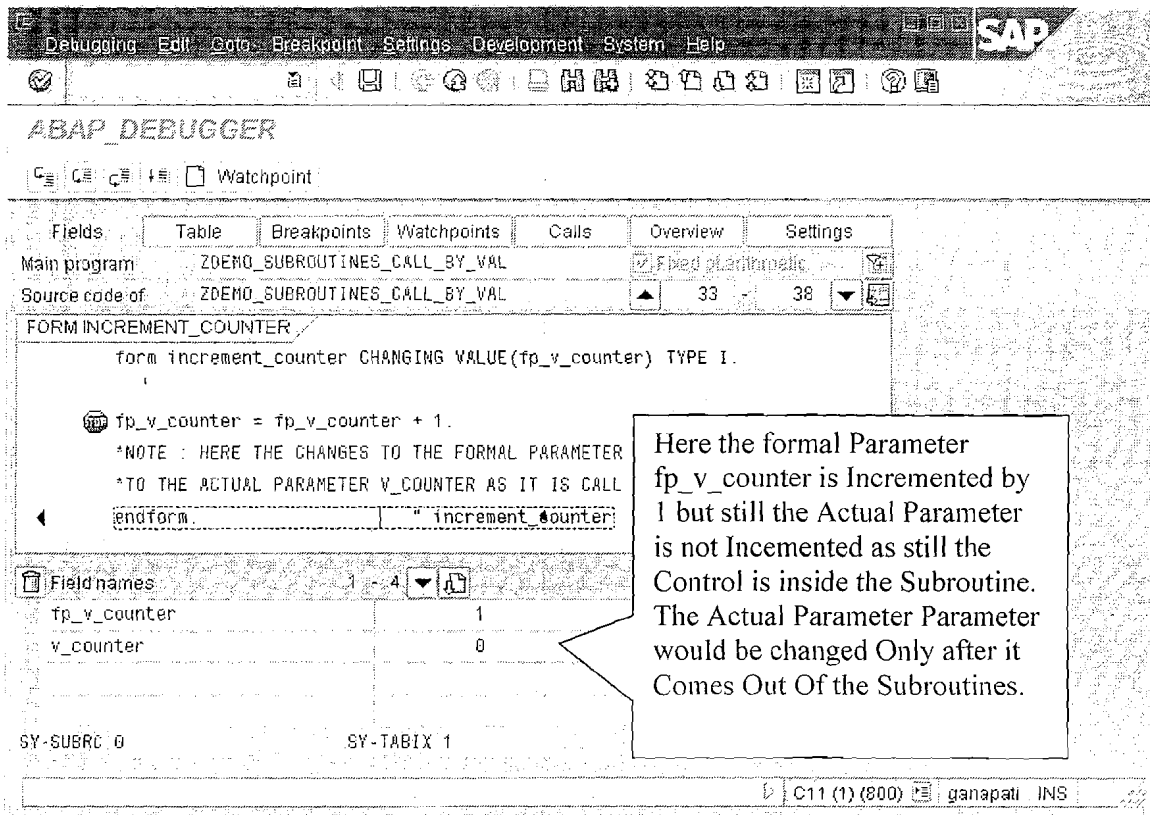
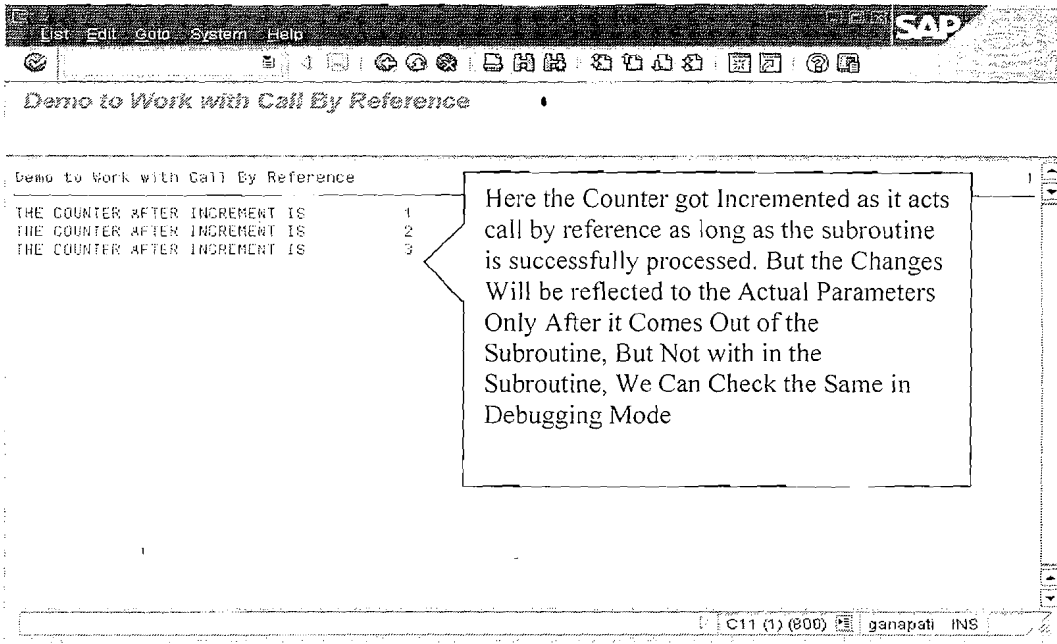
```
form increment_counter CHANGING VALUE(fp_v_counter) TYPE I.
```

```
fp_v_counter = fp_v_counter + 1.
```

*NOTE : HERE THE CHANGES TO THE FORMAL PARAMETER DOESN'T REFLECTS

*TO THE ACTUAL PARAMETER V_COUNTER AS IT IS CALL BY CAL endform. " increment_counter

OUTOUT :



Specifying the Type of Formal Parameters

Formal parameters can have any valid ABAP data type. You can specify the type of a formal parameter, either generically or fully, using the TYPE or LIKE addition. If you specify a generic type, the type of the formal parameter is either partially specified or not specified at all. **Any attributes that are not specified are inherited from the corresponding actual parameter when the subroutine is called.** If you specify the type fully, all of the technical attributes of the formal parameter are defined with the subroutine definition.

By specifying the type, you ensure that a subroutine always works with the correct data type.

Specifying Generic Types

The following types allow you more freedom when using actual parameters. The actual parameter need only have the selection of attributes possessed by the formal parameter. The formal parameter adopts its remaining unnamed attributes from the actual parameter.

Type specification

Check for actual parameters

No type specification

The subroutine accepts actual parameters of any type. The formal parameter inherits all of the technical attributes of the actual parameter.

TYPE ANY

TYPE C, N, P, or X

The subroutine only accepts actual parameters with the type C, N, P, or X. The formal parameter inherits the field length and DECIMALS specification (for type P) from the actual parameter.

TYPE TABLE

The system checks whether the actual parameter is a standard internal table. This is a shortened form of TYPE STANDARD TABLE (see below).

TYPE ANY TABLE

The system checks whether the actual parameter is an internal table. The formal parameter inherits all of the attributes (line type, table type, key) from the actual parameter.

TYPE INDEX TABLE

The system checks whether the actual parameter is an index table (standard or sorted table). The formal parameter inherits all of the attributes (line type, table type, key) from the actual parameter.

TYPE STANDARD TABLE

The system checks whether the actual parameter is a standard internal table. The formal parameter inherits all of the attributes (line type, key) from the actual parameter.

TYPE SORTED TABLE

The system checks whether the actual parameter is a sorted table. The formal parameter inherits all of the attributes (line type, key) from the actual parameter.

TYPE HASHED TABLE

The system checks whether the actual parameter is a hashed

table. The formal parameter inherits all of the attributes (line type, key) from the actual parameter.

Specifying Full Types

When you use the following types, the technical attributes of the formal parameters are fully specified. The technical attributes of the actual parameter must correspond to those of the formal parameter.

Type specification	Technical attributes of the formal parameter
TYPE D, F, I, or T	The formal parameter has the technical attributes of the predefined elementary type
TYPE <type>	The formal parameter has the type <type> This is a data type defined within the program using the TYPES statement, or a type from the ABAP Dictionary
TYPE REF TO <cif>	The formal parameter is a reference variable (ABAP Objects) for the class or interface <cif>
TYPE LINE OF <itab>	The formal parameter has the same type as a line of the internal table <itab> defined using a TYPES statement or defined in the ABAP Dictionary
LIKE <f>	The formal parameter has the same type as an internal data object <f> or structure, or a database table from the ABAP Dictionary

When you use a formal parameter that is fully typed, you can address its attributes statically in the program, since they are recognized in the source code.

The TABLES Addition

To ensure compatibility with previous releases, the following addition is still allowed before the USING and CHANGING additions:

```
FORM <subr> TABLES ... <itab;> [TYPE <t>|LIKE <f>] ...
```

The formal parameters <itab;> are defined as standard internal tables **with header lines**. If you use an internal table without header line as the corresponding actual parameter for a formal parameter of this type, the system creates a **local header line** in the subroutine for the formal parameter. If you pass an internal table with a header line, the table body and the table work area are passed to the subroutine. Formal parameters defined using TABLES cannot be passed by reference. If you want to address the components of structured lines, you must specify the type of the TABLES parameter accordingly.

From Release 3.0, you should use USING or CHANGING instead of the TABLES addition for internal tables, although for performance reasons, you should not pass them by value.

Note : The **sequence** of the actual parameters in the PERFORM statement is crucial. The value of the first actual parameter in the list is passed to the first formal parameter, the second to the second, and so on. The additions USING and CHANGING have exactly the same meaning. You only need to use one or the other. However, for documentary reasons, it is a good idea to divide the parameters in the same way in which they occur in the interface definition.

If a subroutine contains TABLES parameters in its interface, you must specify them in a TABLES addition of the PERFORM statement before the USING and CHANGING parameters. **TABLES parameters are only supported to ensure compatibility with earlier releases, and should no longer be used.**

External subroutine calls

The principal function of subroutines is for modularizing and structuring local programs. However, subroutines can also be called externally from other ABAP programs. In an extreme case, you might have an ABAP program that contained nothing but subroutines. These programs cannot run on their own, but are used by other ABAP programs as pools of external subroutines.

When you call a subroutine externally, you must know the name of the program in which it is defined:

```
PERFORM <sub>(<prog>) [USING ... <p>... ]  
                [CHANGING... <p>... ] [IF FOUND].
```

You specify the program name <prog> statically. You can use the IF FOUND option to prevent a runtime error from occurring if the program <prog> does not contain a subroutine <sub>. In this case, the system simply ignores the PERFORM statement.

When you call an external subroutine, the system loads the whole of the program containing the subroutine into the internal session of the calling program

Data Handling in Subroutines

Global Data from the Main Program

Local Data in the Subroutine

The Parameter Interface

Global Data from the Main Program

Subroutines can access all of the global data in the program in which they are defined (**main program**). You therefore do not need to define a parameter interface if you do not want to change any data in the subroutine, or if very little data is involved.

However, if you want subroutines to perform complex operations on data **without affecting the global data in the program, you should define a parameter interface through which you can pass exactly the data you need**. In the interests of good programming style and encapsulation, you should **always** use a **parameter interface**, at least when the subroutine changes data.

Protecting Global Data Objects against Changes

To prevent the value of a global data object from being changed inside a subroutine, use the following statement:

```
LOCAL <f>.
```

This statement may only occur between the FORM and ENDFORM statements. With LOCAL, you can preserve the values of global data objects which cannot be hidden by a data declaration inside the subroutine.

For example, you cannot declare a table work area that is defined by the TABLES statement with another TABLES statement inside a subroutine. If you want to use the table work area locally, but preserve its contents outside the subroutine, you must use the LOCAL statement.

Local Data in the Subroutine

Data declarations in procedures create local data types and objects that are only visible within that procedure.

Static Local Data Objects

If you want to keep the value of a local data object after exiting the subroutine, you must use the STATICS statement to declare it instead of the DATA statement. With STATICS you declare a data object that is globally defined, but only locally visible from the subroutine in which it is defined.

```
*****
* PROGRAM : ZDEMO_STATICS_IN_SUBROUTINES
* AUTHOR  : GANAPATI . ADIMULAM
* PURPOSE : WRITING SUBROUTINE USING STATICS VARIABLE
*
*****
REPORT ZDEMO_STATICS_IN_SUBROUTINES .

PERFORM increment_counter.
PERFORM increment_counter.

SKIP.
*STATICS Variable is Used in the Subroutine
PERFORM add_number_by_one.
PERFORM add_number_by_one.

FORM increment_counter.

Data : l_counter type i.

L_counter = l_counter + 1.
Write : / 'The Value Of Counter is', l_counter.

ENDFORM.

FORM add_number_by_one.

*STATICS Variable will be Initialized Only Once But Not
*Every time we Call the Subroutine Like Local Variables
STATICS l_number type i.
l_number = l_number + 1.
Write : / 'The Value Of Number is', l_number.

ENDFORM.
```

When you run the program, the following is displayed:

The screenshot shows an SAP IDE editor window with the title "Demo to Work With STATICS". The menu bar includes "List", "Edit", "Goto", "System", and "Help". The toolbar contains various icons for file operations and development. The main text area displays the following output:

```
Demo to Work With STATICS Keyword is  
The Value Of Counter is 1  
The Value Of Counter is 1  
  
The Value Of Number is 1  
The Value Of Number is 2
```

Two callout boxes provide explanations:

- The first callout box points to the first two lines of output and contains the text: "Here Counter is Local Variable, i.e Initialized every time to ZERO, so that it will be 1 always after Each Increment".
- The second callout box points to the last two lines of output and contains the text: "Here Counter is STATIC Local Variable, i.e Initialized OnlyOne time to ZERO, so that it will be Increment by 1 every time we call the Subroutine."

The status bar at the bottom of the window shows "C11 (1) (800) ganapati INS".

Working With Macros :

If you want to reuse the same set of statements more than once in a program, you can include them in a macro. For example, this can be useful for long calculations or complex WRITE statements. **You can only use a macro within the program in which it is defined, and it can only be called in lines of the program following its definition.**

Note : Macros Should be Defined First and Called next in the Program.

SYNTAX to Define the Macro:

DEFINE <Macro Name>.

<Statements>. {Processing Logic through Place Holders i.e &1,&2,...&9}

END-OF-DEFINITION.

You must specify complete statements between DEFINE and END-OF-DEFINITION. These statements can contain up to nine placeholders (&1, &2, ..., &9). You must define the macro **before** the point in the program at which you want to use it.

Macros **do not** belong to the definition part of the program. This means that the DEFINE...END-OF-DEFINITION block is not interpreted before the processing blocks in the program. At the same time, however, macros **are not operational statements** that are executed within a processing block at runtime. When the program is generated, macro definitions are not taken into account at the point at which they are defined.

SYNTAX to Call the Macro :

<Macro Name> [<p1> <p2> ... <p9>].

When the program is generated, the system replaces <macro> by the defined statements and each placeholder &i by the parameter <p_i>.

NOTE: The most essential feature of a macro definition is that it should occur **before** the macro is used.

Requirement : Define a Macro which works for any arithmetic Operation.

DATA: RESULT TYPE I.

PARAMETER : P_INPUT1 TYPE I,
P_INPUT2 TYPE I.

*MACRO DEFINITION
DEFINE OPERATION.

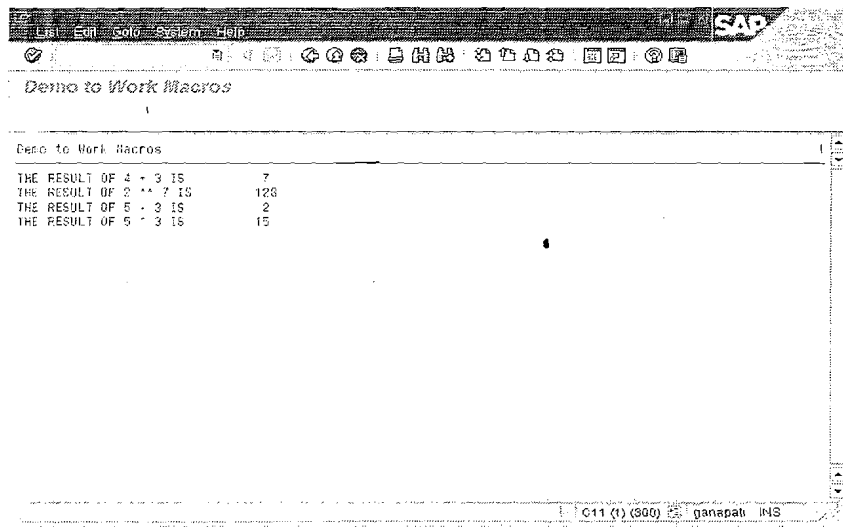
```

RESULT = &1 &2 &3.
WRITE : / 'THE RESULT OF &1 &2 &3.IS', RESULT.
END-OF-DEFINITION.
    
```

```

*CALL THE MACRO OPERATION FOR VARIOUS OPERATIONS
OPERATION 1 + 3.
OPERATION 2 ** 5.
OPERATION P_INPUT1 - P_INPUT2.
    
```

***OUTPUT OF THE PROGRAM :**



Example 2 : Write a Macro to read the First Record from the Given Internal Table.

```

*****
* PROGRAM : ZDEMO_MACRO_READ_FIRST_RECORD *
* AUTHOR : GANAPATI . ADIMULAM *
* PURPOSE : WRITING A MACRO TO READ THE FIRST RECORD *
* FROM THE GIVEN INTERNAL TABLE *
*****
REPORT ZDEMO_MACRO_READ_FIRST_RECORD .
    
```

```

DATA : BEGIN OF WA_BNKA,
      BANKS LIKE BNKA-BANKS, "Bank country key
      BANKL LIKE BNKA-BANKL, "Bank key
      BANKA LIKE BNKA-BANKA, "Name of bank
      STRAS LIKE BNKA-STRAS, "House number and street
      ORT01 LIKE BNKA-ORT01, "CITY
      END OF WA_BNKA.
    
```

DATA IT_BNKA LIKE TABLE OF WA_BNKA .

* START-OF-SELECTION. *

START-OF-SELECTION.

SELECT BANKS "Bank Country Key
BANKL "Bank Key
BANKA "Name of the Bank
STRAS "House & Street No
ORT01 "city
INTO TABLE IT_BNKA
FROM BNKA.

*MACRO TO READ THE FIRST RECORD
DEFINE GET_FIRST_RECORD.

READ TABLE &1 INTO &2 INDEX 1.

END-OF-DEFINITION.

*CALL THE MACRO
GET_FIRST_RECORD IT_BNKA WA_BNKA.

* THE MACRO READS THE FIRST RECORD FROM THE INTERNAL TABLE
* PLACES THE CONTENTS INTO WORK AREA

WRITE : / 'THE REULT OF THE MACRO IS'.
SKIP 1.
WRITE : /5 WA_BNKA-BANKS,
15 WA_BNKA-BANKL,
25 WA_BNKA-BANKA,
40 WA_BNKA-STRAS,
50 WA_BNKA-ORT01.

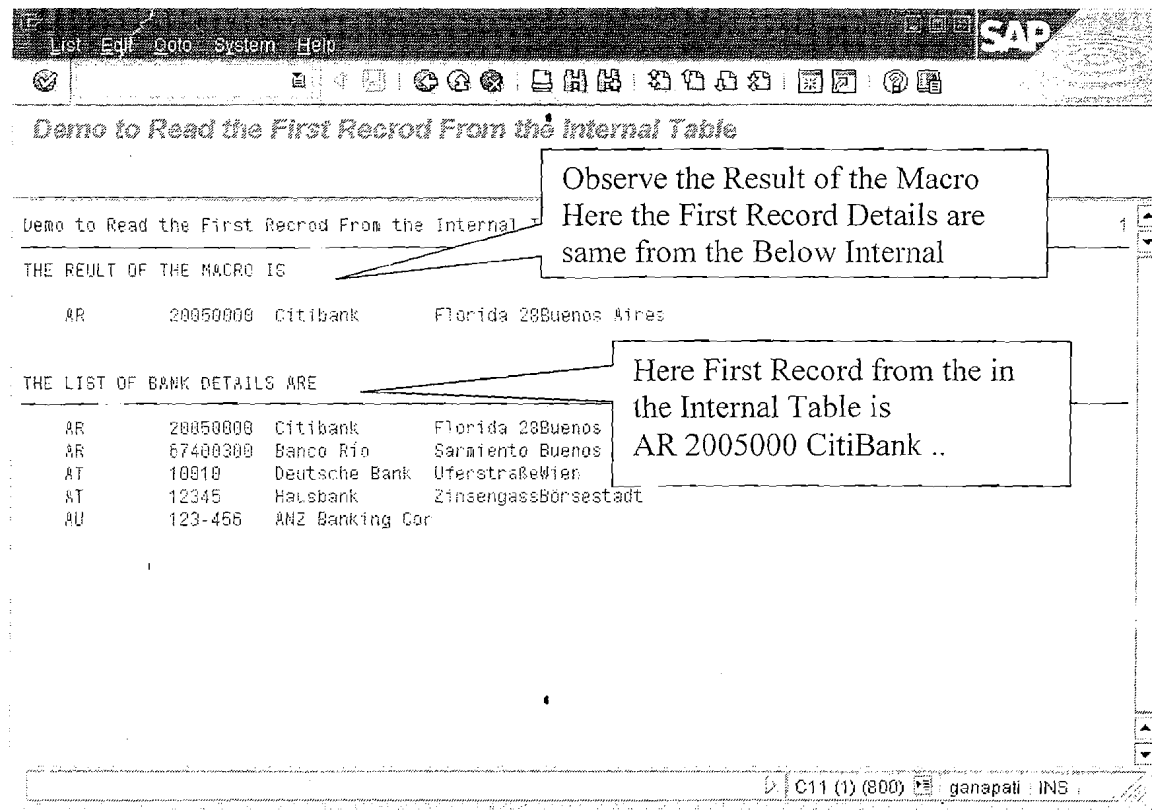
* END-OF-SELECTION. *

SKIP 2.
WRITE : / 'THE LIST OF BANK DETAILS ARE'.
ULINE.

LOOP AT IT_BNKA INTO WA_BNKA FROM 1 TO 5.
WRITE : /5 WA_BNKA-BANKS,

```

15 WA_BNKA-BANKL,
25 WA_BNKA-BANKA,
40 WA_BNKA-STRAS,
50 WA_BNKA-ORT01.
ENDLOOP.
    
```



B) Processing blocks that are called using the ABAP runtime system:

Modularization at Runtime .

- **Event blocks** Ex : INITIALIZATIION, AT SELECTION-SCREEN, START-OF-SELECTION,END-OF- SELECTION.
- **Dialog modules**

Ex : PROCESS BEFORE OUTPUT.(PBO)

PROCESS AFTER INPUT.(PAI)

Exercises

- 1) **Write a Function Module to Get the List Of items and their Details for the Given Purchasing Doc.no?**
- 2) **Write a Function Module to Get the list Of Customers that are created in the Given Date Range?**
- 3) **Write a Function Module to Get the list Of Vendors that are created Only On Saturdays and Sundays in the Given Date Range?**
- 4) **Write a Function Module to Get the list Of Vendors that are created Only On Saturdays and Sundays in the Given Date Range?**
- 5) **Write a Function Module to Get the list materials for the Given Material Type?**
- 6) **Write a Function Module to Get the list of Programs that are Created on Current Date (Today)?**
- 7) **Write a Function Module to Get the list of all the Standard SAP tables ?**
- 8) **Write a FM to Get the list of Sales Orders which Costs More than 1 lakh INR for the Given Data Range ?**
- 9) **Write a FM to Return the list of Storage location under the Given Plant ?**
- 10) **Write a FM to Return the list of Bank Details for the Given Customer?**