Enhancing Finite Capacity Model For Planned Independent Requirements In SAP R/3

P - 2068

A Project Report

Submitted by

C.Ranganayaki - 71205409005

in partial fulfillment for the award of the degree
of

Master of Engineering
in
Industrial Engineering

DEPARTMENT OF MECHANICAL ENGINEERING
KUMARAGURU COLLEGE OF TECHNOLOGY
COIMBATORE – 641 006

ANNA UNIVERSITY :: CHENNAI 600 025

JUNE– 2007

ANNA UNIVERSITY :: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report entitled "**Enhancing Finite Capacity Model for Planned Independent Requirements**" is the bonafide work of

Miss. C.Ranganayaki - Register No. 7120504090005

Who carried out the project work under my supervision.

Signature of the Head of the Department
PROF. N. GUNASEKARAN
PROFESSOR
DEPT. OF MECHANICAL ENGG.
KUMARAGURU COLLEGE OF TECH.
COIMBATORE - 641 006
Ph. (0422) 2609421 (O), (0422) 2653530 (R)
Email gunak_tcbe_in@yahoo.com

Signature of the supervisor
Dr. S. THIRUGNANAM, M.E., Ph.D.
Asst. Professor
Dept of Mechanical Engg.
Kumaraguru College of Technology
Coimbatore-6, India.

Internal Examiner

External Examiner

DEPARTMENT OF MECHANICAL ENGINEERING
KUMARAGURU COLLEGE OF TECHNOLOGY
COIMBATORE 641 006

ROOTS

SL. No. : 1965

Date : 11.05.07

PROJECT / INPLANT TRAINING / INTERNSHIP CERTIFICATE

This is to certify that ~~Mr.~~ / Ms. C. RANGANAYAKI

M.E year student of KUMARAGURU COLLEGE OF TECHNOLOGY has done ~~undergone~~ ~~Project~~ ~~Inplant training~~ / Internship on LIVE PROJECT - ENHANCING FINITE [redacted] FOR PLANNED INDEPENDANT REQUIREMENTS IN SAP in our ROOTS INDUSTRIES LIMITED during the period from NOV -06 to APR -07

During this period ~~his~~ / her conduct was GOOD

(KAVIDASAN)
GENERAL MANAGER - CORPORATE HRD.

KUMARAGURU COLLEGE OF TECHNOLOGY

COIMBATORE - 641 006. TAMILNADU - INDIA

ISO 9001

DEPARTMENT OF MECHANICAL ENGINEERING

National Conference

on

"ADVANCES IN MECHANICAL SCIENCES (AIMS 2007)"

Certificate

This is to certify that Dr. / Mr. / Ms. C.RANGANAYAKI *has participated and presented a paper titled* ENHANCING FINITE CAPACITY MODEL FOR PLANNED INDEPENDENT REQUIREMENTS IN SAP R/3

in the AICTE Sponsored National Conference on "ADVANCES IN MECHANICAL SCIENCES (AIMS 2007)" *during 22 - 24th March 2007.*

C.Velmurugan

Mr.R.Sudhakaran & Mr.C.Velmurugan
Organizing Secretaries

N. Gunasekaran

Dr.N.Gunasekaran
Convener & HOD$_{I/c}$

Dr.Joseph V.Thanikal
Principal

ABSTRACT

Presently companies strive for excellence to become a long term partner. Information Technology plays major role and provides various tools to achieve business excellence. Enterprise Resource Planning (ERP) is one of the tool, which integrates various modules within a single package called SAP R/3 (Systems, Applications and Products in data processing).

Productivity has to be improved by better planning and scheduling of resources to meet the demand of customers. Production Planning is one of the module of SAP R/3 which includes Master Production Schedule (MPS), Material Requirement Planning (MRP), Capacity Planning and scheduling etc. The Capacity Planning follows either Infinite capacity or finite capacity. Most manufacturers embark on finite capacity so that they can make better negotiated, revised delivery promises as per the priority change. The Planned Independent Requirement (PIR) model of production Planning (PP) accepts the demand requirement of customer. The current system follows infinite capacity to accept the demand. The company needs to follow finite capacity to effectively utilize their resources. If the demand requirement is calculated based on infinite capacity model then the bottleneck problem will arise in the production. It increases pressure in the production department. There will be a delay during the delivery of the products. To avoid these problems a software model has been developed to accept the demand, based on finite capacity using ABAP language. This research work deals with the finite capacity at Assembly level, so that the bottleneck problem is avoided at the starting stage itself.

This Project has been successfully implemented in the ROOTS industries, Coimbatore. The Transaction screens were prepared for planned independent requirements. The inputs are stored in database table so that the duplication of data is completely eliminated.

ஆய்வுச் சுருக்கம்.

தற்போது தொழில் துறையில் நிறுவனங்கள் நீண்ட நாள் நிலைத்து நிற்க தரமான சேவையை சவாலாக ஏற்றுள்ளது. தகவல் தொழில்நுட்பம் இதில் முக்கிய பங்கு வகிக்கிறது. என்டர்பிரைஸ் ரிசோர்ஸ் பிளானிங் தகவல் தொழில் நுட்பத்தினை அடிப்படையாக கொண்ட சிறந்த மென்பொருள் ஒரு நிர்வாகத்தின் அனைத்து தகவல்களையும் ஒருங்கிணைந்து சேகரித்து வைக்கும் தன்மையை இந்த மென்பொருள் கொண்டுள்ளது. சாப் R/3 (SAP R/3) என்ற பேக்கேஜ் முதன்மை வாய்ந்தது.

நிர்வாகத்திற்கு தேவையான அடிப்படை வளங்களைத் துல்லியமாக திட்டமிட்டு வழிவகை செய்வதன் மூலம் ஒரு நிர்வாகம் நிறைந்த லாபத்தினை எளிதில் ஈட்ட முடியும். இதற்காக உற்பத்தி திட்டமிடுதல்(PP) என்ற ஒரு தனிப்பகுதியை சாப் R13 கொண்டுள்ளது. நிர்வாகத்தின் அடிப்படை வளங்கள் மற்றும் வடிக்கையாளரின் தேவையை சரி செய்ய நிர்வாகம் இரண்டு வகையான உற்பத்தி முறையை பின்பற்றுகிறது. வாடிக்கையாளரின் தேவைக்கு ஏற்ப அடிப்படை வளங்களை திட்டமிட்டு செயற்படித்துதல் ஒரு வகை. தற்போதுள்ள சாப் R/3 பேக்கேஜ் இந்த முறையை பின்பற்றுகிறது. அடிப்படை வளங்களை பின்பற்றி வாடிக்கையாளரின் தேவையை பூர்த்தி செய்தல் மற்றொரு வகை. சாப் R/3 இந்த முறையை பின்பற்ற இந்த ஆய்வு மேற்கொள்ளப்பட்டுள்ளது. இதன் மூலம் உற்பத்தி நெருக்கடியை நிர்வாகம் ஆரம்ப நிலையிலேயே கட்டுப்படுத்தலாம் மற்றும் வாடிக்கையாளரின் தேவையை எளிதில் பூர்த்தி செய்யலாம்.

இத்திட்டம் ரூட்ஸ் என்ற தனியார் நிறுவனத்தின் உற்பத்தி திட்டமிடுதல் பிரிவில் வெற்றிகரமாக செயல்படுத்தப்பட்டுள்ளது. அனைத்து விதமான இயக்கங்களையும் ஆராய்து உற்பத்தி நெருக்கடியை ஆரம்ப நிலையிலேயே கட்டுப்படுத்த ஒரு முறையை அபாப் என்ற கணிப்பொறி மொழியை பயன்படுத்தி உருவாக்கப்பட்டு, சாப் ஆர்/3 என்ற மென்பொருளில் உள்ள உற்பத்தி திட்டமிடுதல் என்ற பகுதியில் இணைக்கப்பட்டுள்ளது.

ACKNOWLEDGEMENT

The author expresses her sincere thanks to her project supervisor **Dr.S.Thirugnanam**, Assistant Professor of the Department of Mechanical Engineering, Kumaraguru College of technology, Coimbatore for his sustainable interest, valuable and constant guidance and suggestions in carrying out this project successfully in time.

The author shows her sincere thanks to **Dr.N.Gunasekaran**, Head of the Department (i/c) of Mechanical Engineering, Kumaraguru College of Technology, Coimbatore for his encouragement and moral support throughout the M.E. course and especially during the project period.

The author expresses her gratitude to her beloved Principal **Dr.Joseph V.Thanikal**, Kumaraguru College of Technology, Coimbatore for providing an opportunity and necessary facility in carrying out this project work successfully.

The author extends her gratitude to Mr.M.Santhosam, Associate Head-IT and her industry guide Mr.V.Subramaniam, Associate Head-IT Roots industries, Coimbatore for providing valuable guidance and innovative ideas in completing this project.

The author takes this opportunity to thank all the faculty members and technical supporting staffs of the Department of Mechanical Engineering of Kumaraguru College of Technology, Coimbatore for their kind support throughout her course.

Final and foremost the author thanks her lovable parents, family members and friends for their continual support and encouragement throughout her course.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVATIONS

Abbreviations

ABAP	Advanced Business Application Programming
APO	Advanced Planning and Optimization
BAPI	Business Application Programming Interface
ERP	Enterprise Resource Planning
FCS	Finite Capacity scheduling
PIR	Planned Independent Requirements
PP	Production Planning
R/3	Real Time with three tier architecture
SAP	Systems, Applications and Products in Data Processing

Transaction Codes

SE11	ABAP Dictionary
SE14	Adjust and Maintain Database Table
SM37	Database Table Generator
SE38	ABAP Editor
SE51	Screen Painter
SE80	Object Browser
SE93	To create Transaction Code

Chapter 1

Introduction

1.1 Finite Capacity scheduling

Even the best companies tend to be reactive because of poor scheduling. Often their employees prefer being reactive because they are so used to operating in this mode. The shoot-from-the-hip approach is very common in American culture. The West was won with a 'six-shooter' and many shop floor managers continue to function in the shoot-from-the-hip mode. While there are some conditions that might benefit from the shoot -from-the-hip philosophy, modern competitive manufacturing is not one of them from Finite Capacity Scheduling In this groundbreaking book, internationally recognized experts explain why obsolete scheduling methodologies may be preventing your company from achieving competitive levels of productivity, and how today's powerful new finite capacity scheduling (FCS) technologies can transform it into a world-class competitor. The first comprehensive guide to understanding, choosing, implementing, and managing FCS technologies, Finite Capacity Scheduling:

- Exposes the problems inherent to infinite capacity scheduling models.
- Explains why FCS is the key to modern competitive manufacturing.
- Shows how FCS maximizes resource use and cuts inventory costs.
- Proves that FCS is a natural complement to MRP, ERP, TQM, and JIT.
- Reviews current and emerging FCS technologies.
- Offers guidelines for choosing the best FCS system for your company.
- Describes how to integrate FCS seamlessly into your management structure.

Finite capacity scheduling is so-called because it takes capacity into account from the very outset. The schedule is based on the capacity available. Infinite capacity scheduling - the approach used in MRP II - schedules using the customers' order due date and then tries to reconcile the result with the capacity available. There is no single accepted way to carry out Finite Capacity Scheduling, and of the various approaches that exist, some are proprietary secrets.

It is however possible to define certain approaches,

- Electronic scheduling board.
- Order Based Scheduling.
- Constraint based schedulers.
- Discrete Event Simulation.

Finite Capacity Scheduling (FCS) recognizes actual factory capacity limits, including calendars, work centers, labor, and tooling resources. Realistic modeling of capacity limits is the goal of accurate finite capacity scheduling.

Advanced Planning and Scheduling (APS) extends FCS by also recognizing material constraints, in regard to on-hand inventory as well as planned future material deliveries and planned allocations of materials tied to your open Purchase Orders.

1.2 SUMMARY OF PROJECT WORK

1.2.1 Problem Identification

Finite Capacity Scheduling (FCS) is the process of scheduling resources for the production based on finite capacity. The current Production Planning module of the SAP R/3 doesn't allow the FCS. The products are to be produced based on the demand the available in Planned Independent Requirements (PIR).The PIR doesn't restrict the demand when it goes beyond the finite capacity. This Finite Capacity is set by the production department based on resources available. The Existing PIR of SAP R/3 accepts the demand as it is (i.e. it follows infinite capacity scheduling). So there is a possibility for overproduction and bottleneck problem will be arise in the shop floor.

1.2.2 Project Objective

To develop the planned independent requirements with finite capacity using SAP-ABAP /4 to control the demand when it exceeds beyond the finite capacity.

1.2.3 Importance of Project

- Relieves the administration burden.
- Control the Over production.
- Assures the Effective Material Requirement Planning (MRP).
- Assures accurate Budget
- Bottle neck problem is avoided at the starting stage itself.
- No need to maintain inventory
- No need of skilled person to always deal with planned independent requirement

1.2.4 Scope

- No need of skilled persons to deal with the PIR model.
- Reduces the time always to check the demand whether it is going beyond the limit or not.
- Duplication of inputs are avoided.
- Effectively utilize the available SAP resources.
- To prepare transaction screens.

1.3 SEQUENCE OF THE REPORT

The sequence of the entire report goes in the following way:

- Chapter 1 gives an introduction about Finite Capacity Scheduling and summary of the project work. In which problem identification, objective, importance and scope of the project are detailed.

- Chapter 2 deals with SAP and the ABAP language. In ABAP, Module Pool Programming is detailed.

- Chapter 3 focuses on the case study of the company (ROOTS, Coimbatore). It identifies the problem and the methodology to solve it.

- Chapter 4 deals with the step-by-step procedure to develop Planned Independent Requirements with Finite Capacity in SAP R/3.

- Chapter 5 gives the results of the project work being carried out with few transaction screens.

- Chapter 6 concludes the report with the solution for the problem being identified.

1.4 LIMITATIONS

To implement detailed FCS is a tedious process.. After sorting out all the activities from work centers, labor, product lead time, resources the final assembly level demand is calculated. So this project considers the demand only at the final assembly level. If there is any single mistake in data not only in the production planning module, but other module also gets affected To avoid such a complexity, this project considers the demand at the final stage.

1.5 LITRATURE SURVEY

The literature survey is done by referring various journals and websites which has been listed out at the end of the report. The earlier researches carried out by various people mainly focused on the area of ERP. Their survey is revealed the importance of ERP for both manufacturing and service sectors. They listed out various factors for successfully implementing ERP.

The ERP is big software. All the activities of an enterprise are integrated into a single package. So all are argue that ERP installations are difficult to align to a specific requirement of an enterprise. Such a difficulty can be eliminated by bridging the gap between organizational needs and ERP functionality (Collette Rolland and Naveen Prakash,2000).The authors had taken the SAP package in which they consider the module of Material Management .They organize goals and strategies as a directed graph

called map. According to benchmarking done by the partner group in 1998, the installation of each module of ERP system has the following drawbacks.

- High costs
- Difficult and time consuming to implement.
- Difficult alignment to specific requirement of an enterprise
- Need massive allocation of material resources

The detailed FCS has implemented in the new module called APO. To implementation of this module involves investment in time and money. It needs organizational changes, employee acceptance, investment in new software and data collection and so on. By using the proposed method of an enhancing finite capacity model these risks can be reduced significantly.

Enterprise resource planning (ERP) systems are software packages that allow companies to have more real time visibility and control over their operations. There are six common factors (Vidyaranya B. Gargeya and Cydnee Brady, 2005) that are indicative of successful or non-successful SAP implementations. It has been found that the lack of appropriate culture and organizational (internal) readiness as the most important factor contributing to failure of SAP implementations.

The organization expecting that instead buying new module the existing problem can be solved by utilizing facility available in SAP. The proposed model doesn't need any organizational changes it won't take time to implement and the project has developed based on the organization requirement. The organization no need assign skilled person to maintain this model.

Hammer (1990), defined BPR as the "fundamental rethinking and radical redesign of business processes to achieve dramatic improvements in critical measures of performance-cost, quality, capital service and speed". The creative use of IT plays an important role in shaping and restructuring the organization. The author (Fawzy Soliman and Mohamed A. Youssef) explains the role SAP in Business

Process Re-engineering. They have shown ranking given to SAP by Top ten companies. The ranking details are given below.

Business	Percentage of top companies in the world using SAPR/3
Pharmaceutical	70
Petroleum	70
Chemical	100
Computers	87.5

Source: Adapted from *Fortune Magazine* survey (1997)

TABLE 1.1: Percentage of top companies in the world using SAPR/3

Performance measure	Percentage of the top ten companies in the world using SAP/R3
Top Ten companies	70
Most profitable	90

Source: Adapted from *Fortune Magazine* survey (1997)

TABLE 1.2: Percentage on the basis of performance of top companies in the world using SAPR/3.

Chapter 2

SAP R/3 System

2.1 INTRODUCTION TO ERP

ERP is a package with the techniques and concepts for the integrated management of business as a whole, for effective use of management resources, to improve the efficiency of an enterprise. Initially, ERP was targeted for manufacturing industry mainly for planning and managing core business like production and financial market. As the growth and merits of ERP package, ERP software was designed for basic process of a company from manufacturing to small shops with a target of integrating information across the company.

"Enterprise Resource Planning (ERP), is a software driven business management system which integrates all facets of the business, including planning, manufacturing, sales, and marketing".

The different types of ERP are SAP, BAAN, JD Edwards, Oracle Financials, Siebel, and PeopleSoft. Among all the ERP's most of the companies implemented or trying to implement SAP because of number of advantages aver other ERP packages.

2.1.1 Evolution of ERP

The history of ERP can be traced back to the 1960's, when the focus of systems was mainly towards inventory control. Most of the systems software were designed to handle inventory based in traditional inventory concepts. The 1970's witnessed a shift of focus towards MRP (Material Requirement Planning). This system helped in translating the master production schedule into requirements for individual units like sub assemblies, components and other raw material planning and procurement. This system was involved mainly in planning the raw material requirements.

Then, in 1980's came the concept of MRP-II i.e., the Manufacturing Resource Planning which involved optimizing the entire plant production process. Though MRP-II, in the beginning was an extension of MRP to include shop floor and distribution management activities, during later years, MRP-II was further extended to include areas like Finance, Human Resource, Engineering, Project Management etc. This gave birth to

ERP (Enterprise Resource Planning) which covered the cross-functional coordination and integration in support of the production process. The ERP as compared to its ancestors included the entire range of a company's activities.

ERP addresses both system requirements and technology aspects including client/server distributed architecture, RDBMS, object oriented programming etc.

2.1.2 Evaluation Criteria

Some important points to be kept in mind while evaluating ERP software include:

- Functional fit with the Company's business processes
- Degree of integration between the various components of the ERP system
- Flexibility and scalability
- User friendliness
- Ease of implementation
- Ability to support multi-site planning and control
- Technology - client/server capabilities, database independence, security
- Availability of regular upgrades
- Amount of customization required
- Local support infrastructure
- Reputation and sustainability of the ERP vendor
- Total costs, including cost of license, training, implementation, maintenance, customization and hardware requirements.

2.2 OVERVIEW OF SAP

SAP is an enterprise resource planning (ERP) software product capable of integrating multiple business applications, with each application representing a specific business area. These applications update and process transactions in real time mode. It has the ability to be configured to meets the needs of the business.

SAP is the name of the company founded in 1972 under the German name (Systems, Applications, and Products in Data Processing) is the leading ERP (Enterprise Resource Planning) software package.

SAP R/3 is a third generation set of highly integrated software modules that performs common business function based on multinational leading practice. It takes care of any enterprise however diverse in operation, spread over the world. It is a 3-tier architecture system as shown in the Figure 3.1 and it works on the client-server model.

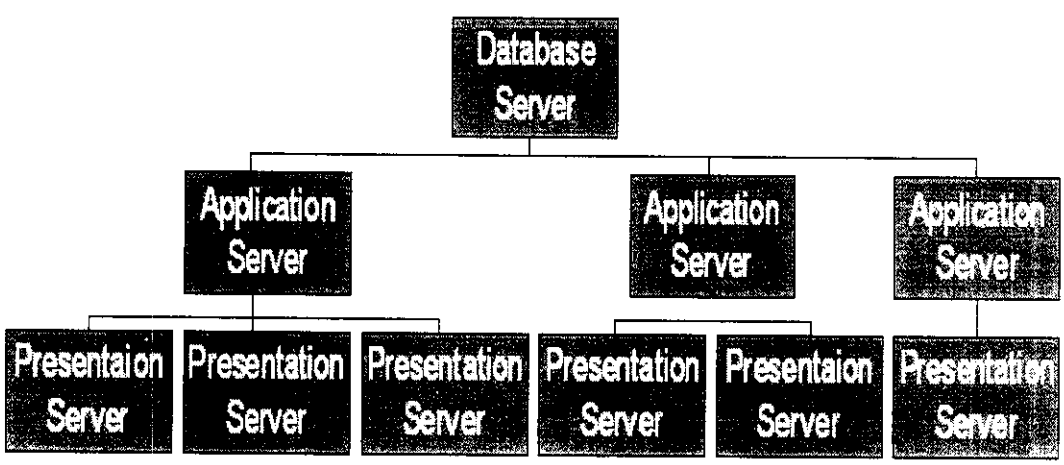


FIGURE 2.1: SAP R/3 ARCHITECTURE

Database Server : Stores Database.

Application Server : Contains Software for Running the Application Logic.

Presentation Server : Graphical User Interface

K C T LIBRARY Coimbatore
P - 2063

In R/3 system all the three servers like presentation, application server and database server are located at different system. The application layer of an R/3 System is made up of the application servers and the message server. Application programs in an R/3 System are run on application servers. The application servers communicate with the presentation components, the database, and also with each other, using the message server. All the data are stored in a centralized server. This server is called database server.

The advantage of having 3 tier or 3 layered Architecture is to reduce traffic on the Database Server. Based on the required degree of distribution, you can combine two servers or all three servers in the same machine (Stand alone).

2.2.1 SAP R/3 Structure

- Application Layer for Standard and new applications.
- Middle ware Layer also called as R/3 Basis.
- Operating System, Database

O.S	Database	GUI	Protocols
NT	Oracle	Win 3.1	TCP/IP
Unix	Ingress	NT, 95	CPC
AS/400	Informix, DB/2		

TABLE 2.1: SAP R/3 STRUCTURE DETAILS

The different types of data are specified as Meta Data: Data that describes the structure of data or Meta Objects is called Metadata. Master Data: Master data is data that remains unchanged over a long period of time. It contains information that is always needed in the same way. With master data you are dealing with attributes, texts or hierarchies. Transaction data: Data relating to the day-to-day transactions.

2.2.2 Functional Modules

SAP is categorized into 3 core functional areas which is common to any type of industry:

- Logistics
  - Sales and Distribution (SD)
  - Material Management (MM)
  - Warehouse Management (WM)
  - Production Planning (PP)
  - General Logistics (LO)
  - Quality Management (QM)
- Financial
  - Financial Accounting (FI)
  - Controlling (CO)
  - Enterprise Controlling (EC)
  - Investment Management (IM)
  - Treasury (TR)
- Human Resources
  - Personnel Administration (PA)
  - Personnel Development (PD)

2.2.3 Technical Module

ABAP/4 stands for Advanced Business Application Programming is the programming language used for the thousand of tiny embedded programs called transactions that make up the SAP application. It is an event driven language. It is the Central part of Middle ware layer that eliminates dependencies from Hardware, Operating Systems or database management systems. ABAP programs are interpreted not complied.

2.3 REASONS TO IMPLEMENT SAP

There are number of technical reasons numbers of companies are planning to implement SAP. It's highly configurable, highly secure data handling, min data redundancy, max data consistency, you can capitalize on economics of sales like purchasing, tight integration-cross function.

- Client server system.
- Multi Lingual.
- It can be easily customized based on country's needs.
- It has OSS (Online Support System)
- It has other services like Information service and Maintenance service.
- It has Preventive service (early watch service).

2.4 LIMITATIONS OF SAP

- Interfaces are huge problem.
- Determine where master data resides.
- Expensive.
- Very complex.
- Demands highly trained staff.
- Lengthy implementation time.

2.5 THE ABAP LANGUAGE

- Advanced Business Application Programming
- Developed by SAP for the interactive development of application programs
- 4th Generation Language (with some OO features)
- Main uses for ABAP programs include
  - creation of new reports
  - development of new user dialog programs (transactions)
  - customization of R/3 to meet individual client needs

2.5.1 Objectives

To introduce

- the ABAP integrated development environment
- the two main ABAP application types
  - reporting
  - dialog programming

2.5.2 ABAP Workbench Purpose & Function

- SAP's Integrated Graphical Programming Development Environment:
  - Used to create/change ABAP application programs
  - Each ABAP application program is either a report or a transaction:
    1. Reports are applications that retrieve and display information from database with little or no user interaction
    2. Transactions accepts inputs/data from users and then perform one or more relevant actions, usually involving updating databases

- The Workbench can be used to
  - write ABAP code
  - design dialogs/screens with a graphical editor
  - create menus with a menu editor
  - debug an application
  - test an application for efficiency
  - control access to objects under development
  - create new or access predefined database information

- Comprises the following tools
  - The Object Navigator (Repository Browser)
  - the ABAP Language
    - ❖ the Screen and Menu Painters

- the Repository Information System
- the Data Modeler
- various test and analysis tools
- the ABAP Query
- the Workbench Organizer

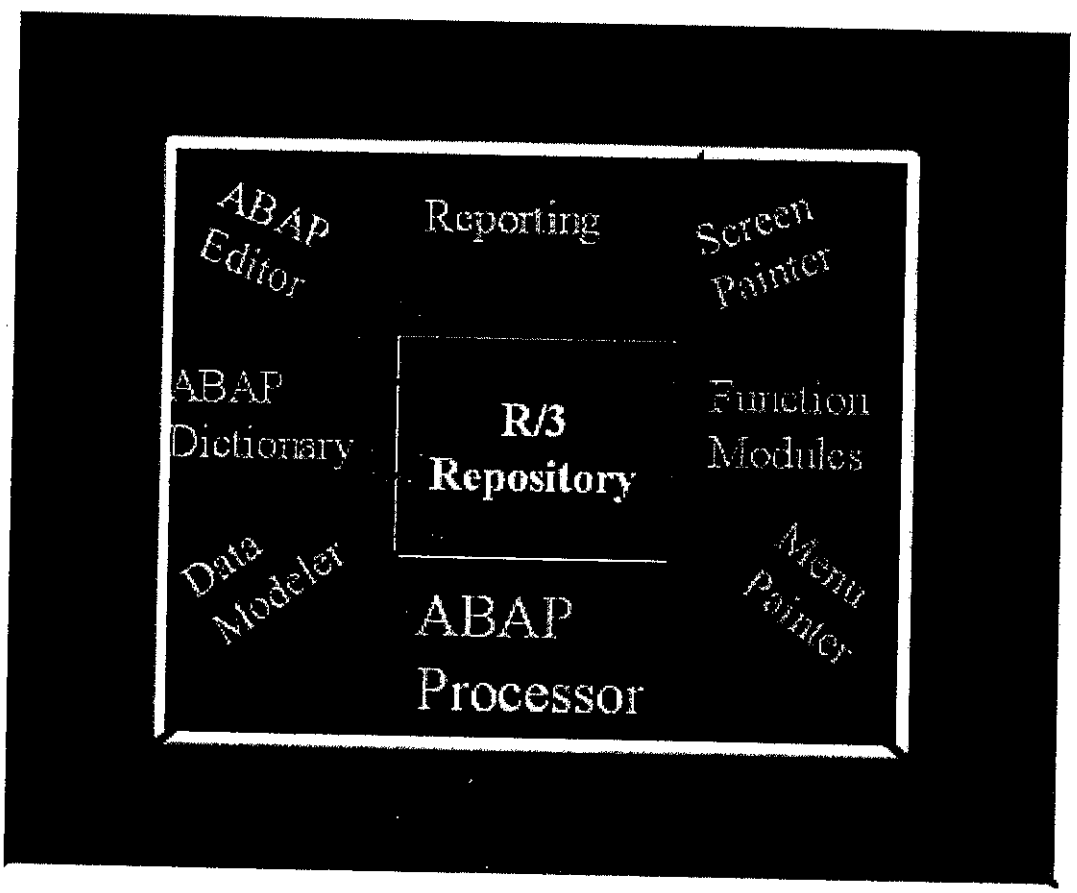


FIGURE 2.2: ABAP WORKBENCH ARCHITECTURE

Application Modules are all written in ABAP/4, which is interpreted by Basis executables, which in turn, run on the operating system. The sole purpose of an R/3 system is to provide a suite of tightly integrated, large-scale business applications. R/3 is the system in which ABAP/4 programs will run. Figure 3.2 shows the architecture of ABAP Workbench.

2.6 OBJECT NAVIGATOR (The Repository Browser)

- central tool for organizing and managing your personal development objects
- used to navigate through development object lists
  - where development objects are the components used to construct an application
- automatically calls other workbench tools
  - if you create a new data definition the browser calls the Data Dictionary, and then returns to the browser after the definition is created.

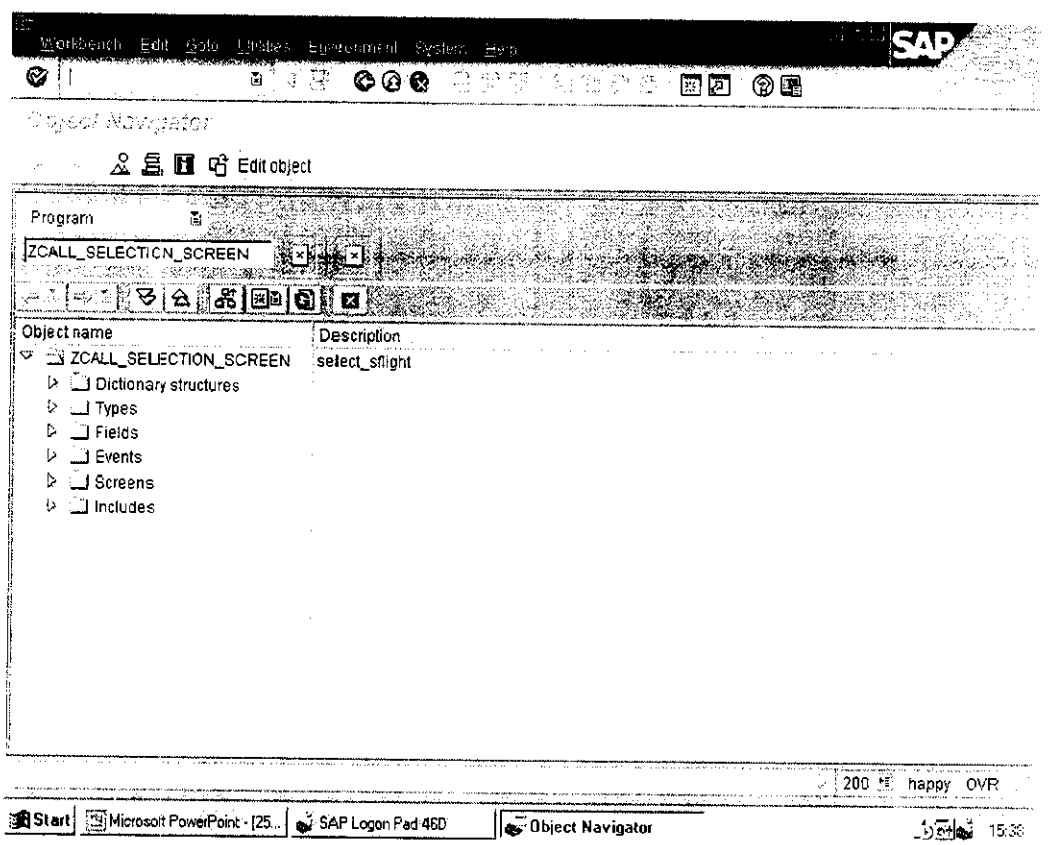


FIGURE 2.3: REPOSITORY BROWSER

2.7 DIALOG PROGRAMMING

- Dialog Programs (*Transactions*) are used for both reading and changing the database
- Main Components of a Dialog Program are
  - screen & menus (as shown in Figure 2.6)
  - processing logic defined in an ABAP program (*module pool*)
  - for each screen, interaction and control flow from one module to another is defined in DYNPRO (Figure2.5)
  - data structures defined in the ABAP dictionary

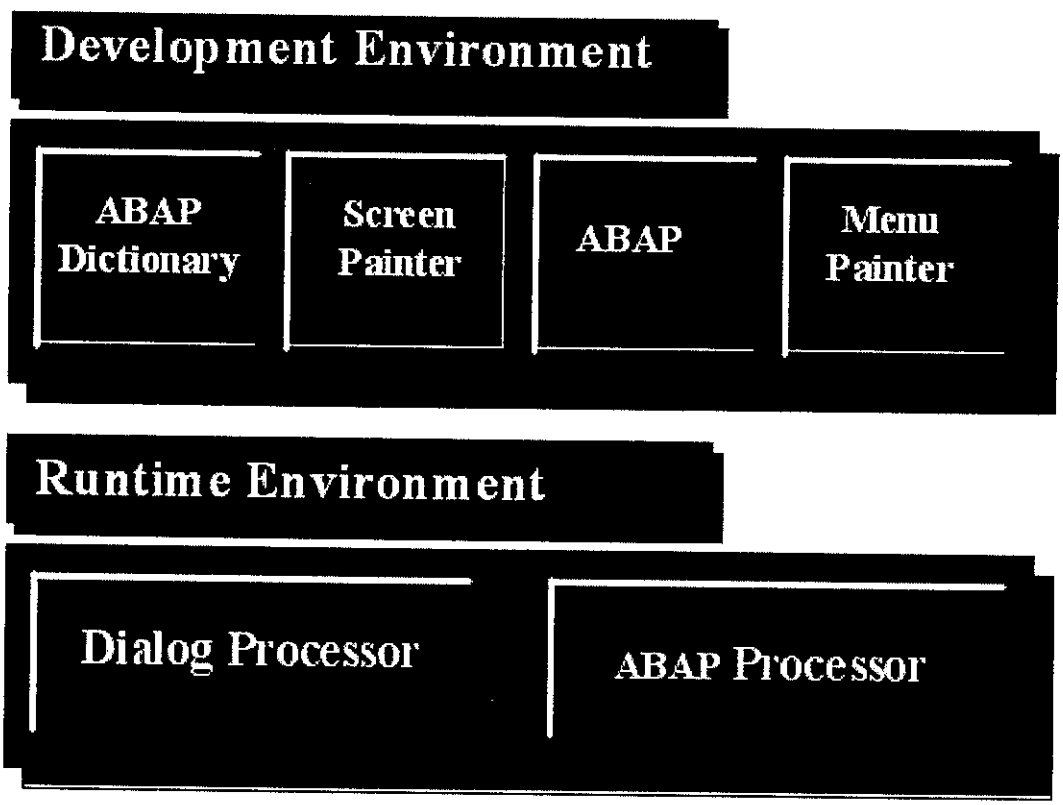


FIGURE 2.4: BLOCK DIAGRAM

Figure 2.4 gives the block diagram of Development environment and Runtime environment of ABAP in SAP R/3 Systems.

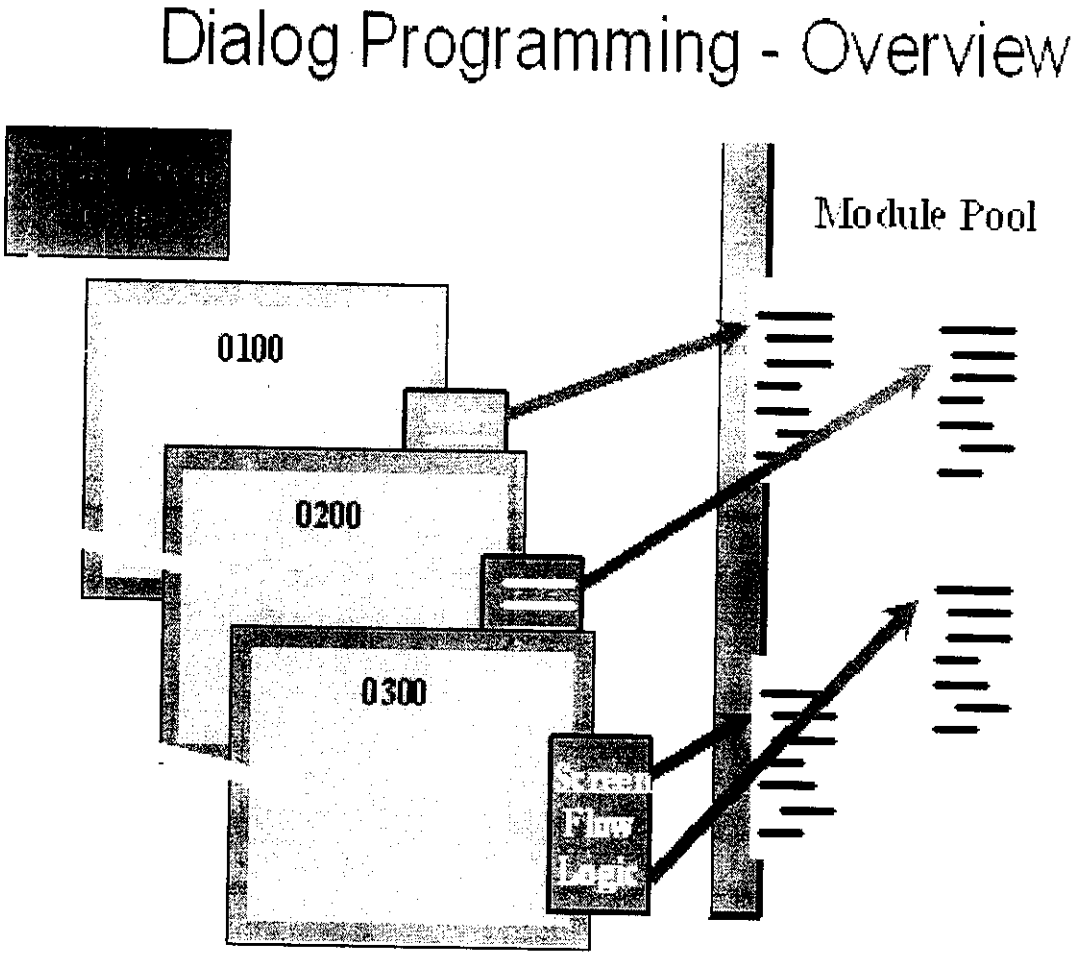


FIGURE 2.5: LINK BETWEEN TRANSACTION CODE AND MODULE POOL

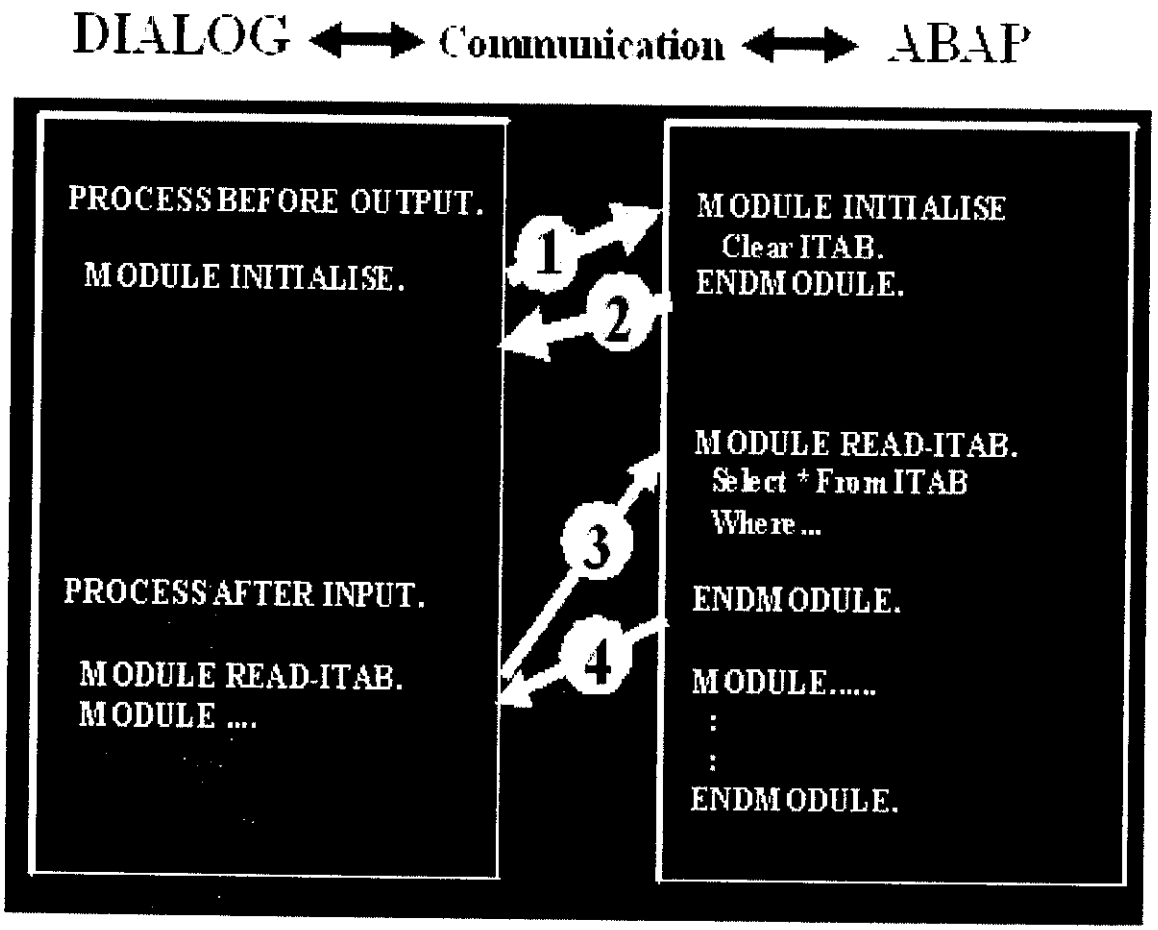


FIGURE 2.6: WORK FLOW IN DIALOG PROGRAMMING

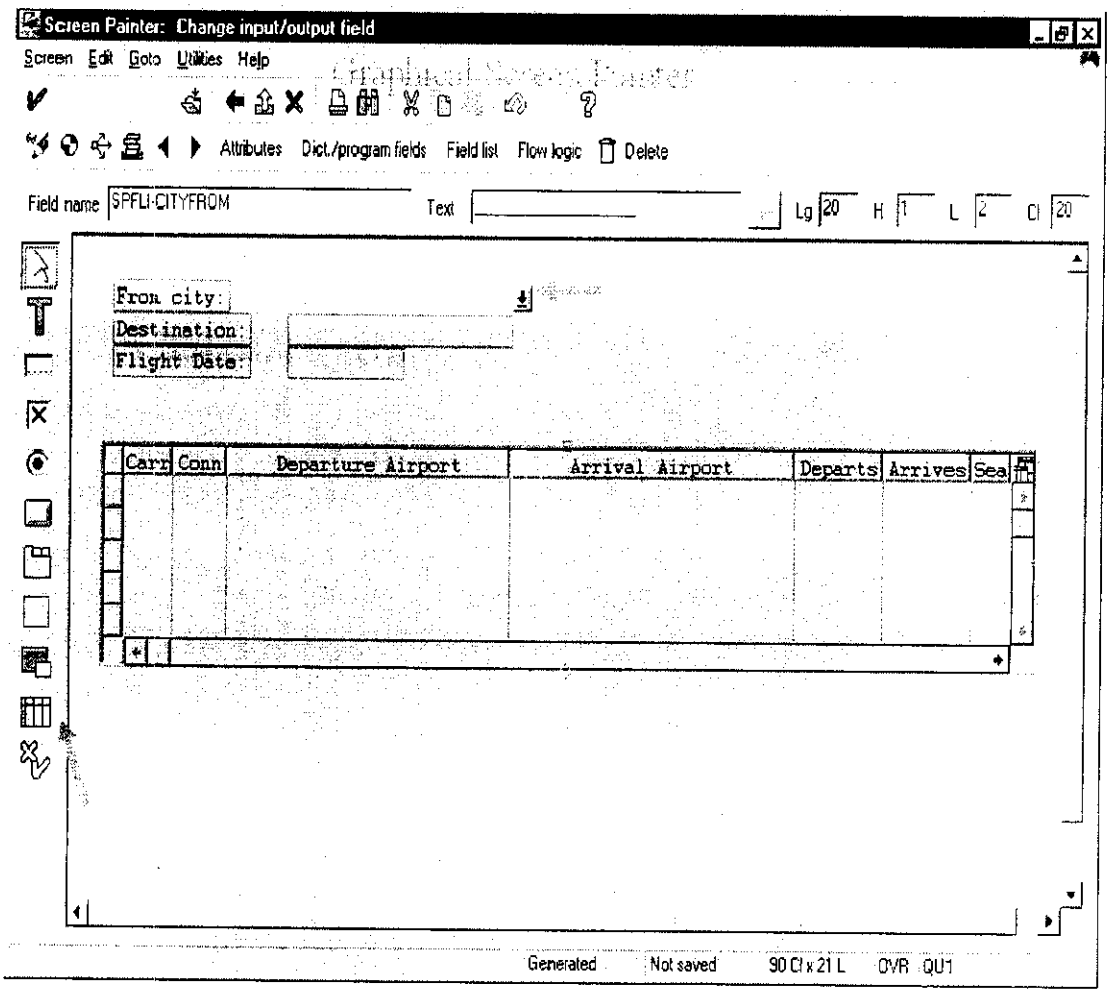


FIGURE 2.7: SCREEN PAINTER

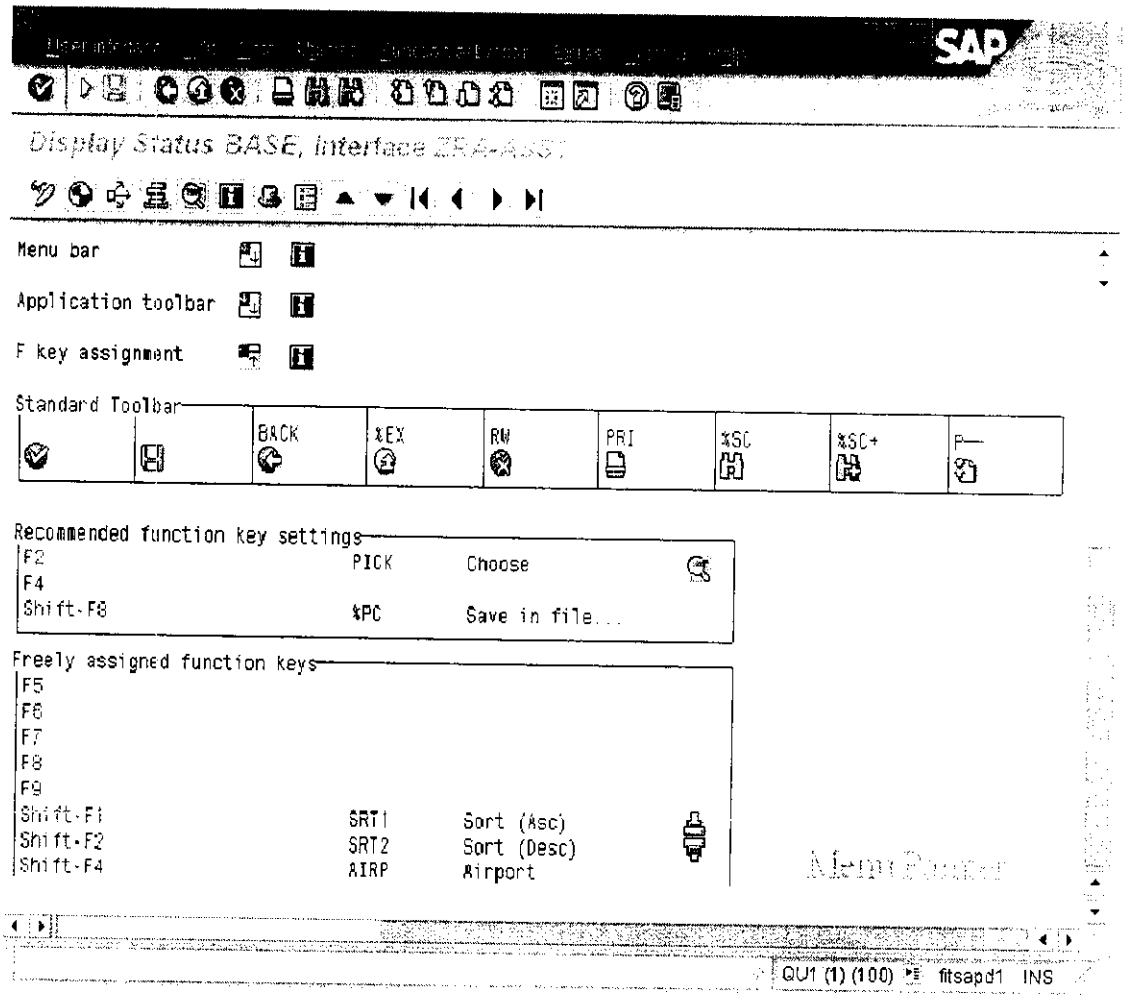


FIGURE 2.8: MENU PAINTER

Chapter 3

Case Study In ROOTS

3.1 COMPANY PROFILE – ROOTS INDUSTRIES LIMITED

3.1.1 Corporate Information

In a dynamic world that is driven by technology, a successful presence depends on the way you mould that technology to fit popular needs. Indigenous talent, a daring attitude, courage to accept and learn new things... and the simple spark of an idea. That is the genesis of ROOTS.

Roots' single minded pursuit of enhancing the quality of life has led to many other diversifications. Roots, today, is a multifaceted corporate entity with interests in automobile accessories, cleaning equipment, castings, precision tools, hi-tech engineering services, healthcare and education.

ROOTS Industries Ltd. is a leading manufacturer of HORNS in India and the Eleventh largest manufacturing company in the world. Headquartered in Coimbatore - India, ROOTS has been a dominant player in the manufacture of Horns and other products like Castings and Industrial Cleaning Machines. Since its establishment in 1970, ROOTS has had a vision and commitment to produce and deliver quality products adhering to International Standards.

With a strong innovative base and commitment to Quality, Roots Industries Limited has occupied a key position in both international and domestic market as suppliers to leading OEMs and after market. Similar to products, Roots has leading edge over competitors on strong quality system base. Now, RIL is the first Indian Company and first horn manufacturing company in the world to get ISO/TS 16949 certification based on effective implementation of QS 9000 and VDA 6.1 system requirement earlier. RIL has entered into technical collaboration with Robert Bosch, SA to further enhance the technical competence. Roots' vision is to become a world class company manufacturing world class product, excelling in human relation.

3.1.2 ROOTS Vision

Roots industries stand technologically ahead of others to deliver world-class innovative products useful to our customers. We will rather lose our business than our customers' satisfaction. It is our aim that the customer should get the best value for his money.

Every member of our company will have decent living standards. We care deeply for our families, for our environment and our society. We promise to pay back in full measure to the society by way of selfless and unstinted service.

3.1.3 Alliences

Roots is a leading Original Equipment supplier to major vehicle manufacturers like Mercedez Benz, Mitsubishi, Mahindra & Mahindra, Toyota, Fiat, TELCO, TVS, Kinetic, etc. The technical collaboration with **Robert Bosch S.A. of Spain** starting from 1995 has strengthened the R&D activities and increased Roots' technical competence to international standards

Roots Multiclean Ltd. (RMCL) is a joint venture with **Hako Werke GmbH & Co., Germany**, one of the largest cleaning machine manufacturers with global operations. RMCL is the sole representative in India and SAARC countries for Hako Werke's entire range of cleaning equipment. The quality of RMCL products is so well established that Hako buys back a major portion for their global market.

3.1.4 Products

- Electric Horns
- Air Horns
- Cleaning Machines
- Castings
- Precision Products
- Metrology
- Poly Products

- Nature Cure Home
- Matriculation School
- Cleaning Solutions.

3.1.5 ROOTS Group

ROOTS Industries Limited	Electric Horns
ROOTS Auto Products Pvt Ltd	Air Horns, Switches & Controllers
ROOTS Multiclean limited	Cleaning Machines
Cleaning Machines	Aluminium & Zinc Pressure Die Cast
Roots Precision Products	Dies, Tools, Jigs & Fixtures
Roots Digital Engineering Service Pvt .Ltd	Digital Engineering Services
Roots Metrology Laboratory	Instrument Calibration, Quality System, Consultancy
Roots Polycraft	Plastic components
R.K Nature Cure Home	Nature Cure Therapy, Yoga & Massages
Satchidananda Jothi Nikethan	International School
Crystal Clean Care	Range of Modern Cleaning Techniques
Roots Industries Malaysia Sdn. Bhd.	Electric Horns

TABLE 3.1 : ROOTS GROUP OF INDUSTRIES

3.1.6 Exports

Roots products have successfully made their presence heard loud and clear in the global market. Roots horns are exported to over 15 countries worldwide. A major share of the exports goes to USA, Japan, Middle East and South America. Roots is the only Indian company that meets the **demanding** standards of the **Japanese markets**. Roots cleaning equipment and die cast parts, etc. are exported to USA, Europe, Australia, Japan, Far East, South America and several other advanced countries.

3.2 EXISTIING SYSTEM IN SAP

Finite scheduling is so called because it considers your capacity to be finite, most manufacturers embark on finite scheduling so they can make better delivery promises to their customers, and better negotiate revised delivery promises as priorities change. The current planned independent requirement in SAP system doesn't have the control when the demand goes beyond the fixed capacity. The demand is accepted as like i.e. it follows infinite capacity model. Advanced Planning Optimization (APO) is another module of SAP in which the PIR always maintain the demand within finite capacity.

3.3 A RECORD OF THE DEMAND PROGRAM IN SAP

The demand program contains the requirement quantities and dates for main assemblies and finished products in the form of planned independent requirements. The requirements ascertainment horizon is used in a sales-order oriented planning scenario. In the Demand Planning stage, the planner in cooperation with the customer enters a demand for the product at customer level. Consequently the customer is required to confirm by placing a firm order within the requirements ascertainment horizon. This is entered into the system as a sales order. After the period has expired, only the demand (planned independent requirements) for which sales orders exist is converted into production orders.

3.4 DRAWBACKS

- No control when demand exceeds beyond the capacity.
- Bottleneck problem in production.
- Less efficiency with the usage of finite capacity model.
- Needs skilled person to check whether the demand is within the capacity or not.
- For implementing new module called APO, they have to invest money and time.
- Inventory problem is also arises.

3.5 PROPOSED METHODOLOGY

To avoid these drawbacks in existing system, a new model has been developed. This model controls the demand, when it exceeds beyond the Finite Capacity level. If the demand exceeds beyond the Finite Capacity level, it will restrict the user to enter the demand of another product. This will extend till the user enters the correct demand value. So the pressure in the Production department is completely avoided in the starting itself. Also there is no duplication of inputs. To avoid Duplication of inputs the foreign key and primary key has used in this model.

In the proposed model the transaction screens were designed These Transaction screens are user friendly. So no need of skilled person to always deal with the Planned Independent Requirements, also no need to check demand with the Finite Capacity, It will automatically done by system when user enters the demand.

3.6 STEP-BY-STEP PROCEDURE

The following are the steps to be completed before starting with the project. which support for the project as back end..

1. Creation of Database Table.
2. Creation of Transaction Screens.
3. Creation of Transaction Code.

The detailed algorithms for the creation of costing system are discussed below.

3.6.1 Creation of database table

Algorithm:

Step 1: Go to SE11 transaction.

Step 2: Select database table and give the table name.

Step 3: Click on create option.

Step 4: Give short description.

Step 5: Select the data browser, type of table view maintenance and delivery class.

Step 6: Select the required fields and give the fieldname.

Step 7: Specify the data elements.

Step 8: Select the primary key.

Step 9: Set the technical settings and the data class.

Step10: Click on Save, Check and Activate.

3.6.2 Creating data type globally in dictionary

Algorithm:

Step 1: Go to SE11 transaction.

Step 2: Click on data type.

Step 3: Give the data type name.

Step 4: Click on create.

Step 5: Select the data element.

Step 6: Click ok.

Step 7: Give the short description.

Step 8: Select the built-in-type option in elementary types.

Step 9: Select the type of data from the available list.

Step10: Specify the length of the data.

Step11: Select the field label.

Step12: Specify the length of the field label.

Step13: Click on Save, Check and Activate

3.6.3 Entering data into the Table

Algorithm:

Step 1: In Menu bar, go to utilities.

Step 2: Select table contents.

Step 3: Select create entries.

Step 4: Give the data.

Step 5: Save the entry.

Step 6: Click on reset.

Step 7: Give the next data.

Step 8: Save the entry.

Step 9: Repeat the process till all the data are entered.

Step10: Click on Back and Save.

3.6.4 Viewing data in the Table

Algorithm:

Step 1: In Menu bar, go to utilities.

Step 2: Go to table contents.

Step 3: Click on display.

Step 4: Press F8 for execution.

Step 5: Select the box to edit the data.

Step 6: Select the table entry.

Step 7: Click on change.

3.6.5 Creation of Transaction Screens

Algorithm:

Step 1: Go to SE80 transaction.

Step 2: Click on edit object.

Step 3: Click on program and give a program name.

Step 4: Click on create option.

Step 5: Press ok.

Step 6: Remove the stars and give a top include program name.

Step 7: Press ok.

Step 8: Give the title.

Step 9: Select the type module.

Step10: Click on Save.

Step11: Click on back.

Step12: Click on edit object.

Step13: Select the program.

Step14: Click on screen and give the screen number.

Step15: Click on create option.

Step16: Give short description.

Step17: Click on layout.

Step18: Prepare the screen on screen painter.

Step19: Click on Save, Check and Activate.

Step20: Go back.

Step21: Click on element list.

Step22: Give some name for ok-code.

Step23: Click on flow logic.

Step24: Remove the comments by deleting the stars.

Step25: Click on Save.

Step26: Double Click on Module user_command_0100.

Step27: Click on yes.

Step28: Click on the main program name.

Step29: Press ok.

Step30: Click on top include.

Step31: Double Click on the top include program name.

Step32: Click on Save, Check and Activate.

Step33: Go back.

Step34: Click on Save, Check and Activate.

Step35: Go back.

Step36: Again click on Save, Check and Activate.

3.6.6 Preparation of Screens on Screen Painter

Algorithm:

Step 1: Select the required type of field from the options available.

Step 2: Give the label name for each field and the text to be typed on the label.

Step 3: Give the appropriate text name.

Step 4: Drag and place the necessary fields from the graphical elements toolbar.

Step 5: Go back.

Step 6: Click on Save, Check and Activate.

3.6.7 In Layout to take the table fields from Dictionary

Algorithm:

Step 1: Click on layout.

Step 2: In go to option, select the secondary window.

Step 3: Click on dictionary/program fields.

Step 4: Give the table name.

Step 5: Click on the option, get from dictionary.

Step 6: Select the fields by clicking left corner button.

Step 7: Click ok.

Step 8: Drag and place it on the screen.

3.6.8 Creation of Transaction Code

Algorithm:

Step 1: Go to SE93 transaction.

Step 2: Give the transaction code name.

Step 3: Click on create.

Step 4: Give the short description.

Step 5: Click ok.

Step 6: Give the program name.

Step 7: Give the screen number.

Step 8: Click on Save.

3.7 NECESSARY REQUIREMENT

These are the requirements which should validated into the project.

- The demand should be accepted for the current year.

- According to the year and month the calendar will be displayed in the screen.
- The model has to restrict the user when demand goes beyond the control till the demand has to be corrected.
- There should be summation for all the demands.
- The duplicated value should not be accepted.
- According to the year and month the demand value are to be updated in the database table.
- The table control should be empty whenever the user enters new month and year.
- There should be options with insert, delete, next page, previous page, select all and deselect all.

Chapter 4

Results

RESULTS & DISCUSSIONS

The Transaction Screens were prepared for the planned independent requirements with the finite capacity model. These screens were validated using sample inputs. All the necessary requirements of the company are satisfied in this research work. The input given by the user is stored in the database at back end. The screen is developed for day format.

Thus, the duplication of inputs is completely eliminated by using foreign key and Primary key, which is one of the main requirements of the company. In the screen, user has to enter the year and month for entering demand in the planned independent requirements form. The demand is to be cancelled are modified according to the customer needs in the screen.

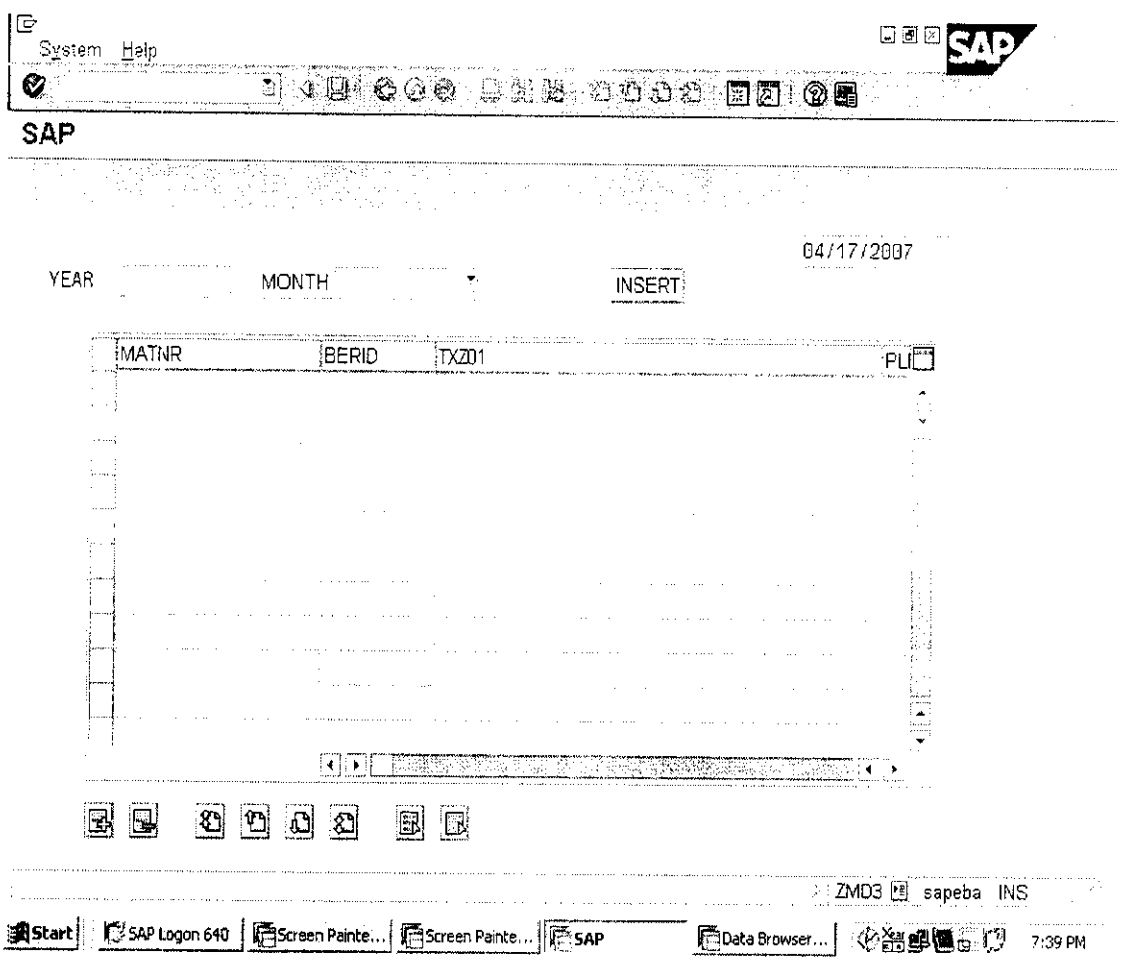


FIGURE 4.1: TRANSACTION SCREEN – Initial screen

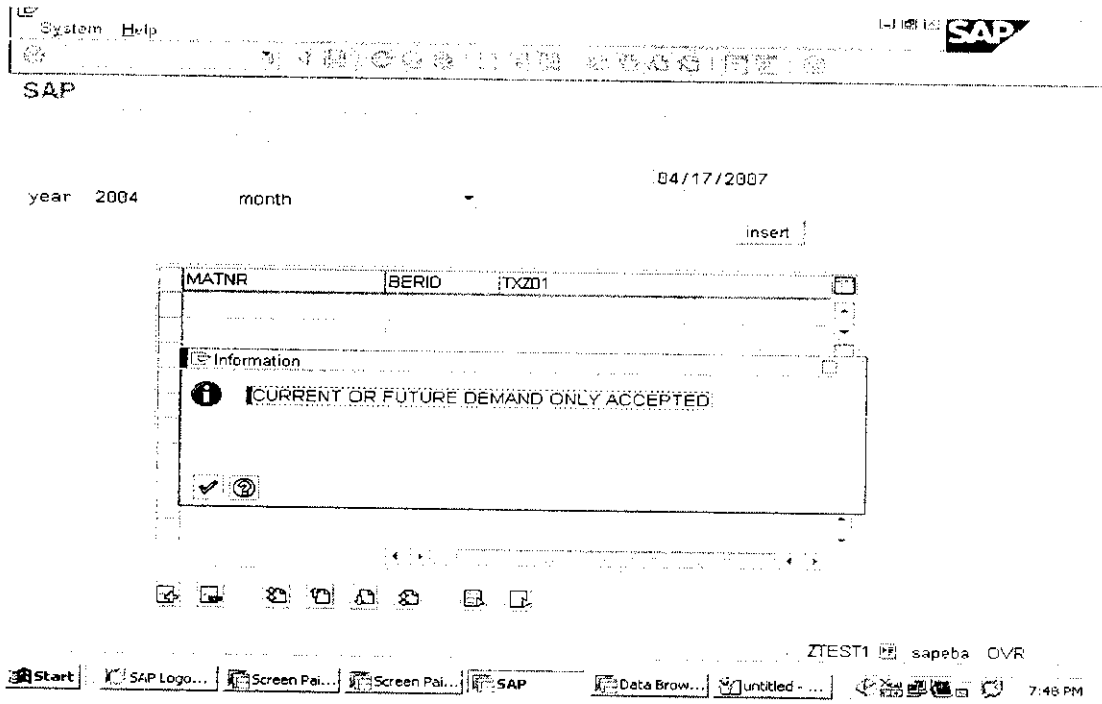


FIGURE 4.2: TRANSACTION SCREEN – information displayed when user enters past year.

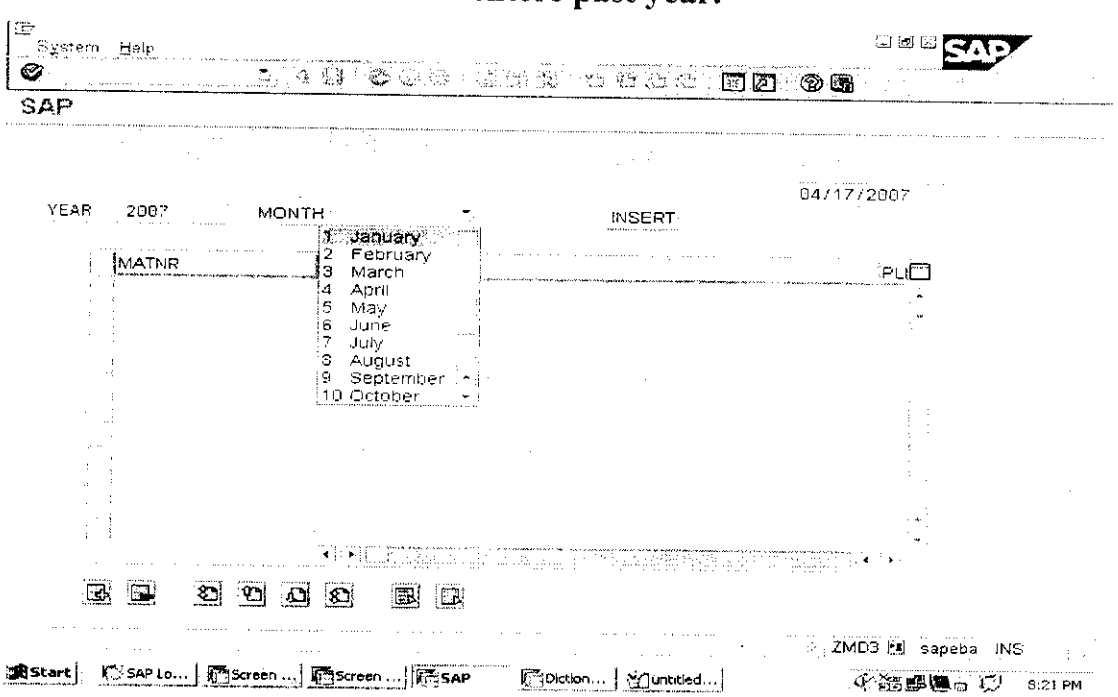


FIGURE 4.3: TRANSACTION SCREEN- List of months

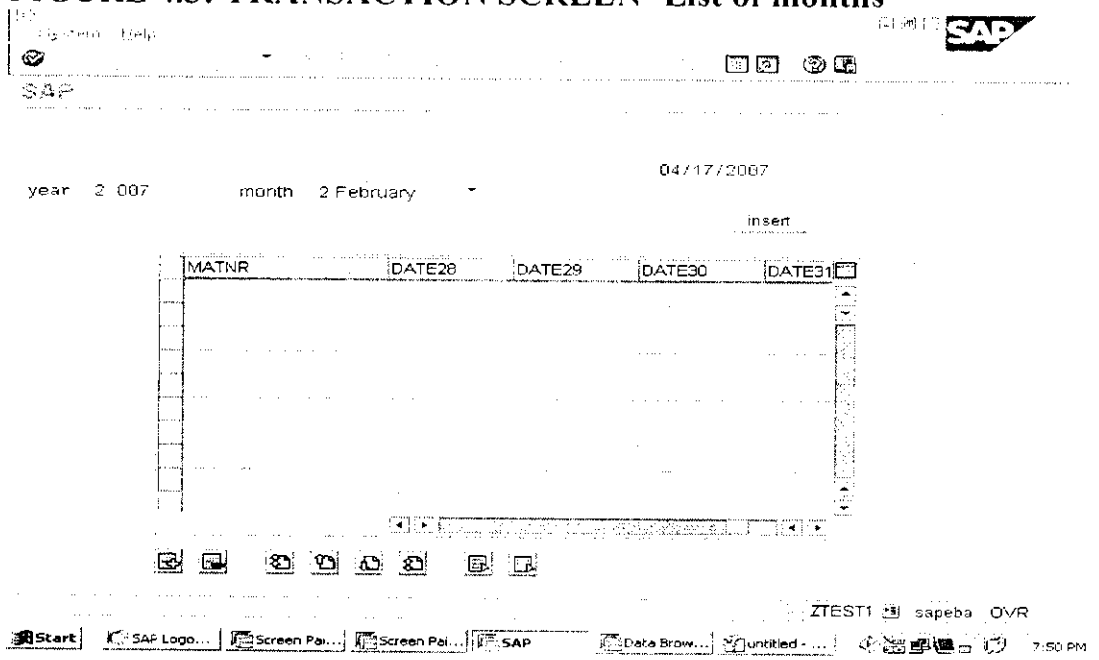


FIGURE 4.4: TRANSACTION SCREEN- For Calendar

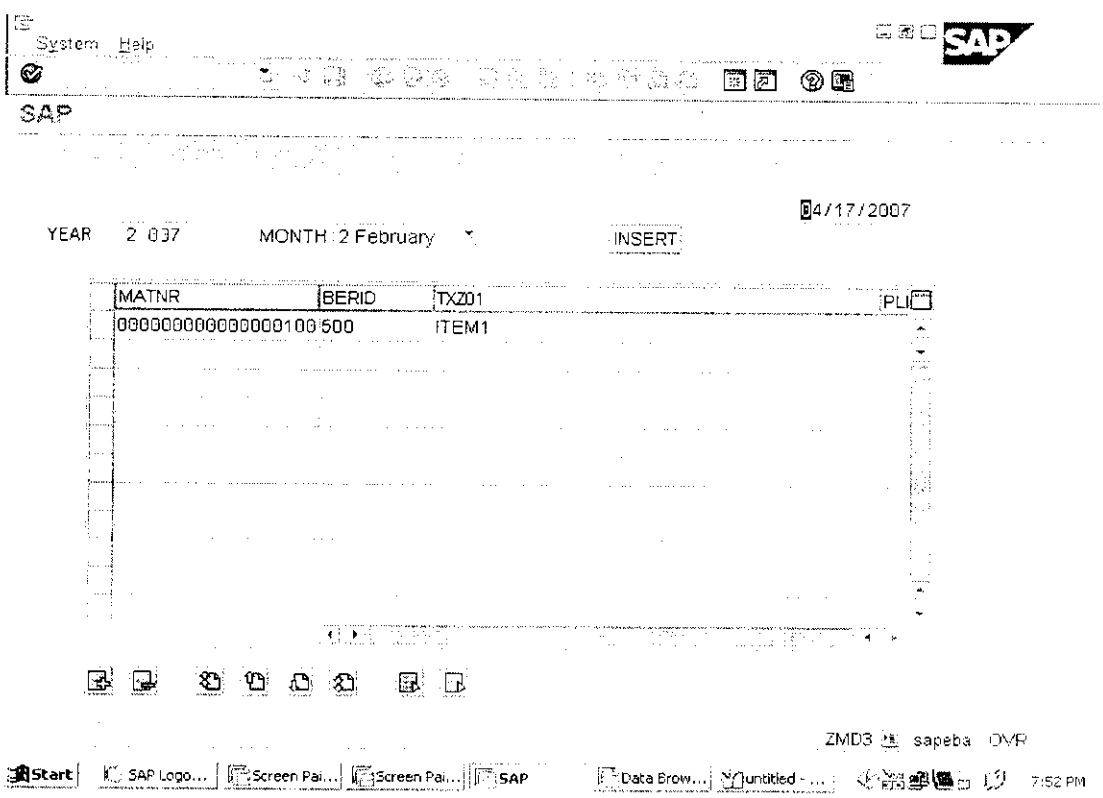


FIGURE 4.5: TRANSACTION SCREEN- Material details

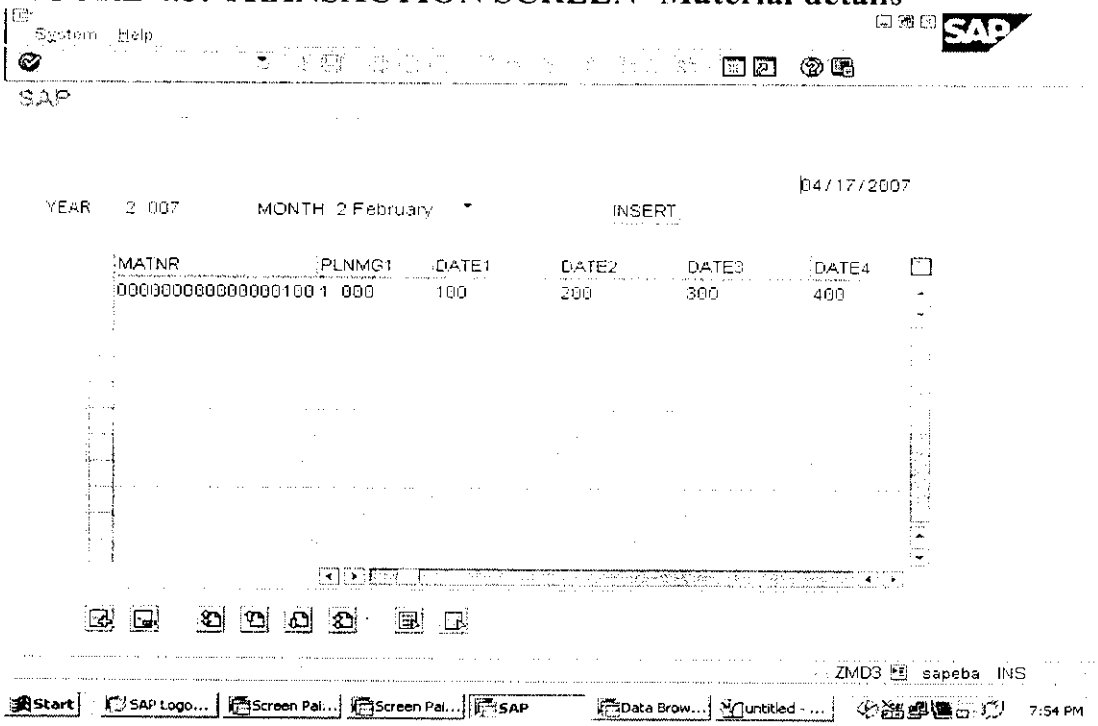


FIGURE 4.6: TRANSACTION SCREEN- Adding Demand

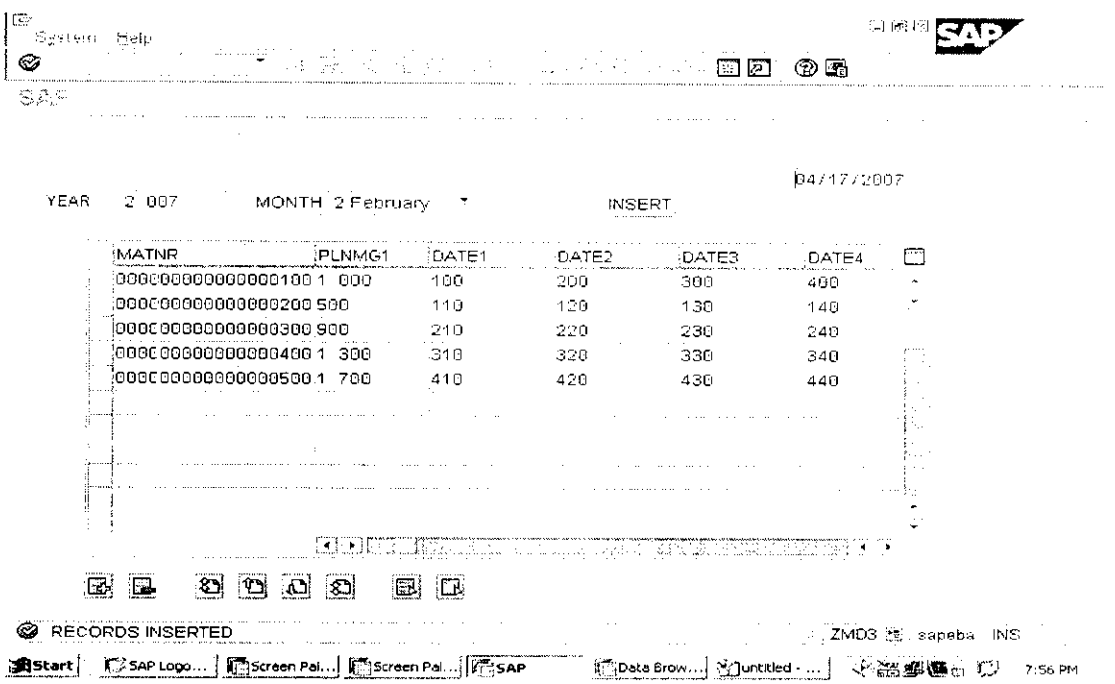


FIGURE 4.7: TRANSACTION SCREEN- PIR Screen

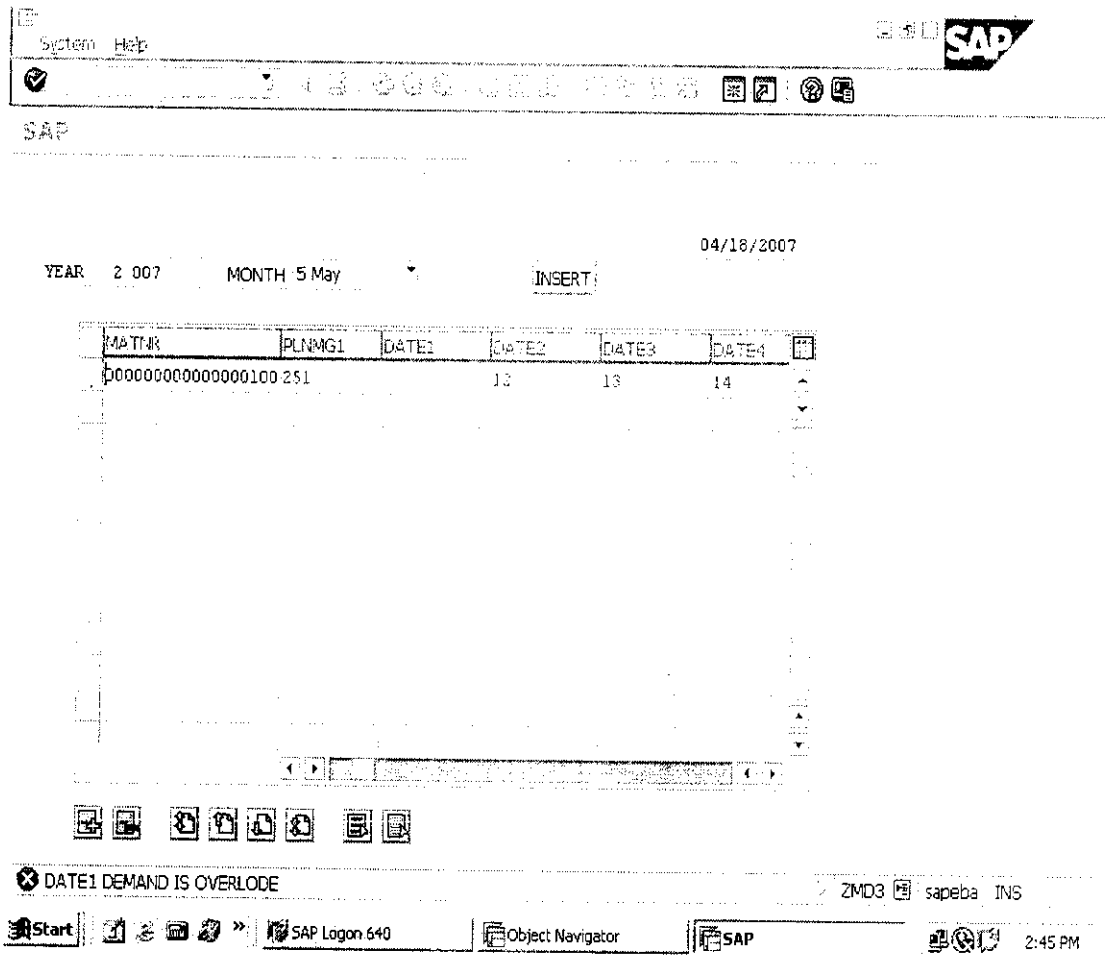


FIGURE 4.8: TRANSACTION SCREEN- The Screen will restrict the user when demand goes beyond the finite capacity

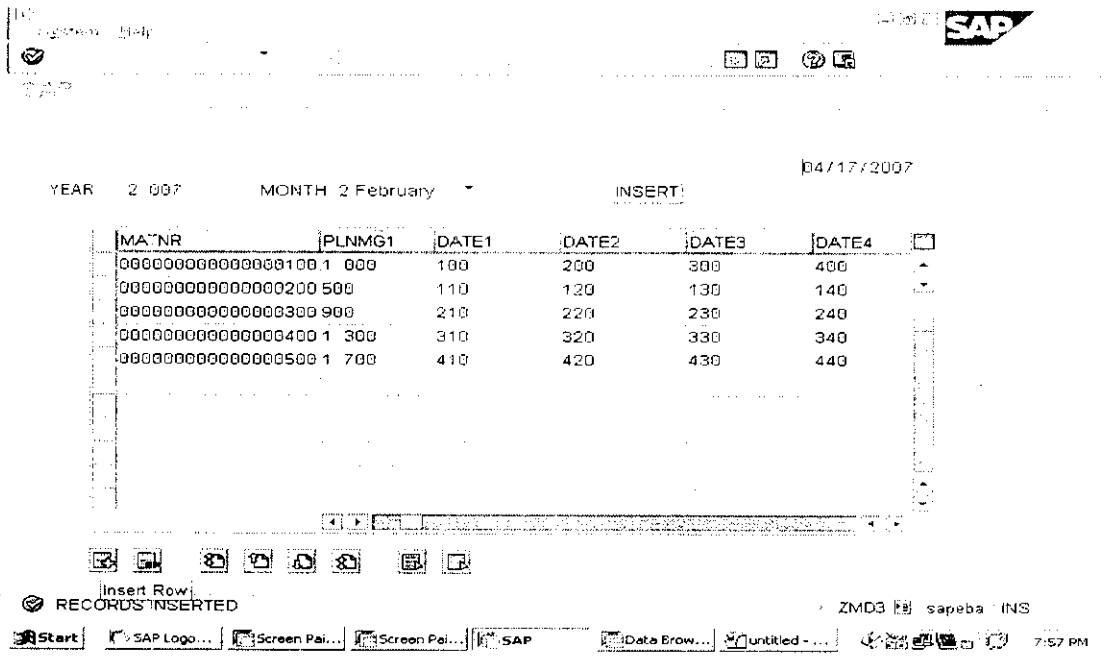


FIGURE 4.9: TRANSACTION SCREEN- Insert Row

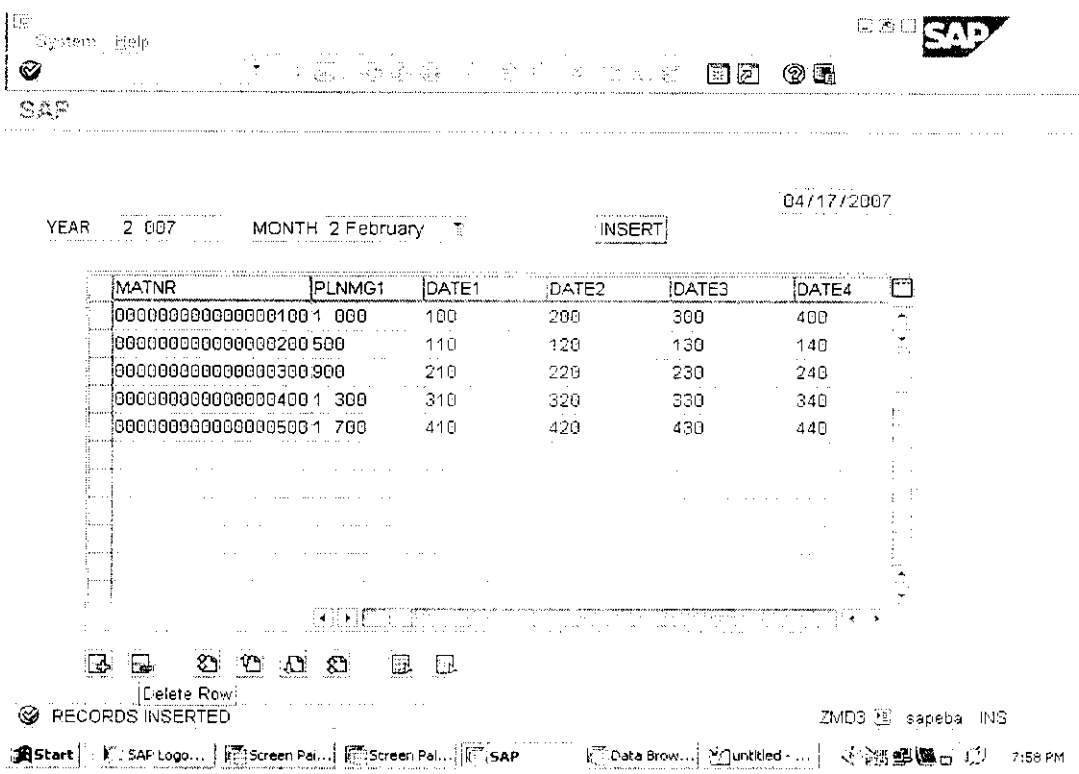


FIGURE 4.10: TRANSACTION SCREEN- Deleting Row

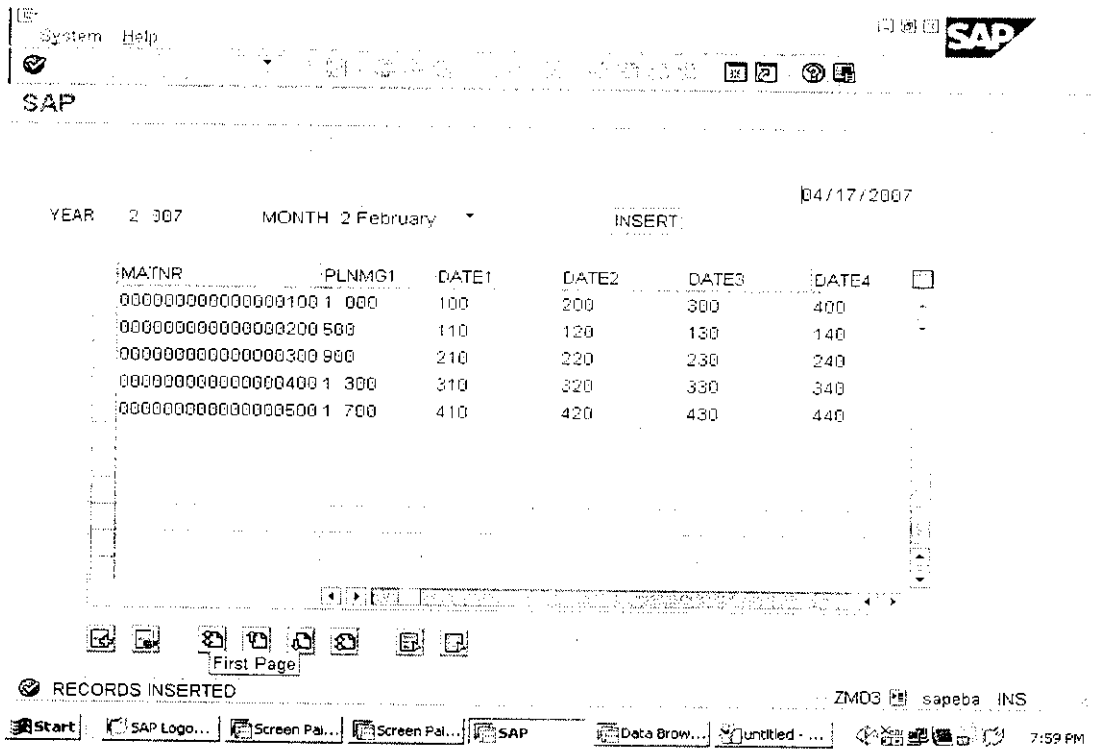


FIGURE 4.11: TRANSACTION SCREEN- showing first page

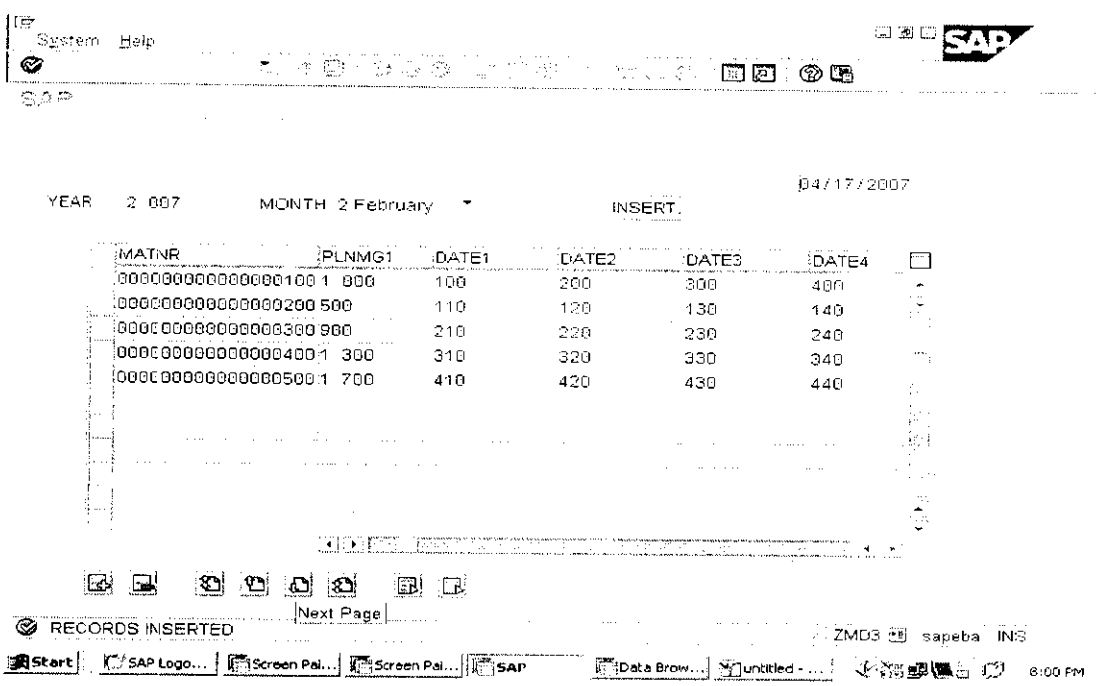


FIGURE 4.12: TRANSACTION SCREEN- showing next page

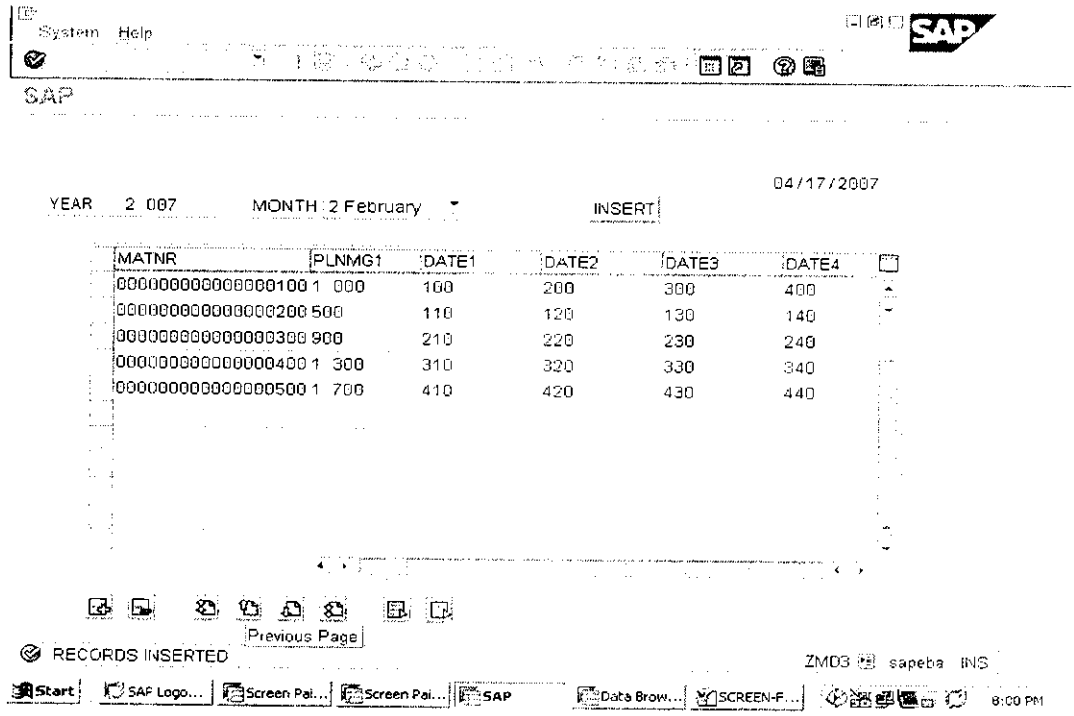


FIGURE 4.13: TRANSACTION SCREEN- showing previous page

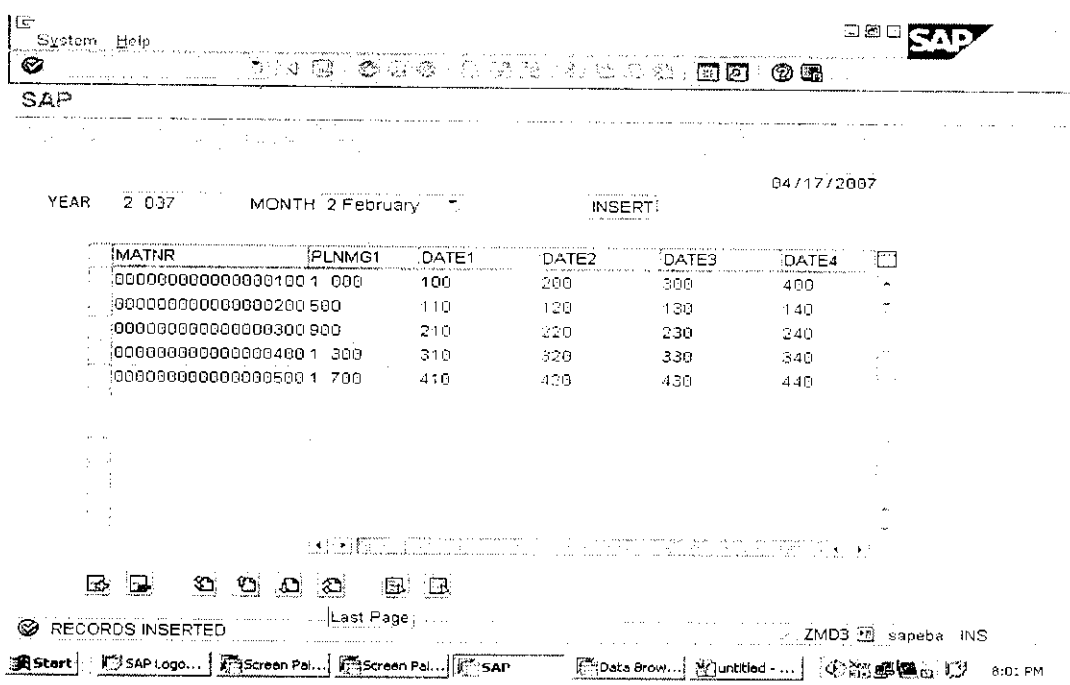


FIGURE 4.14: TRANSACTION SCREEN- showing last page

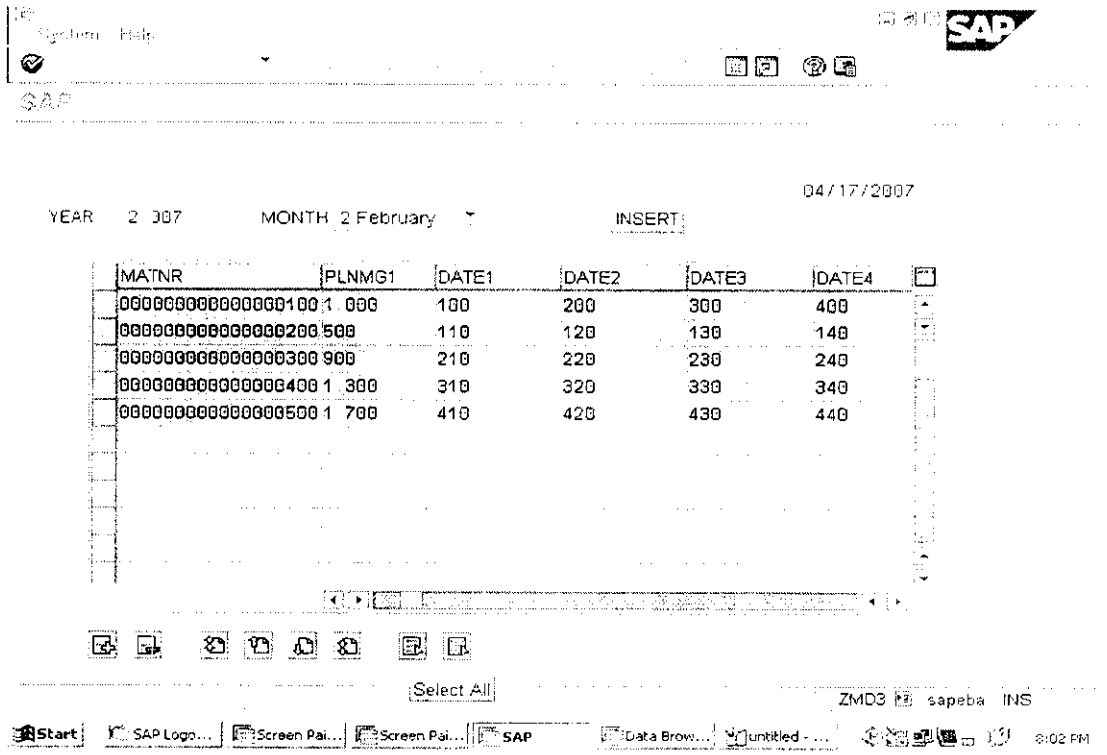


FIGURE 4.15: TRANSACTION SCREEN- showing select all details

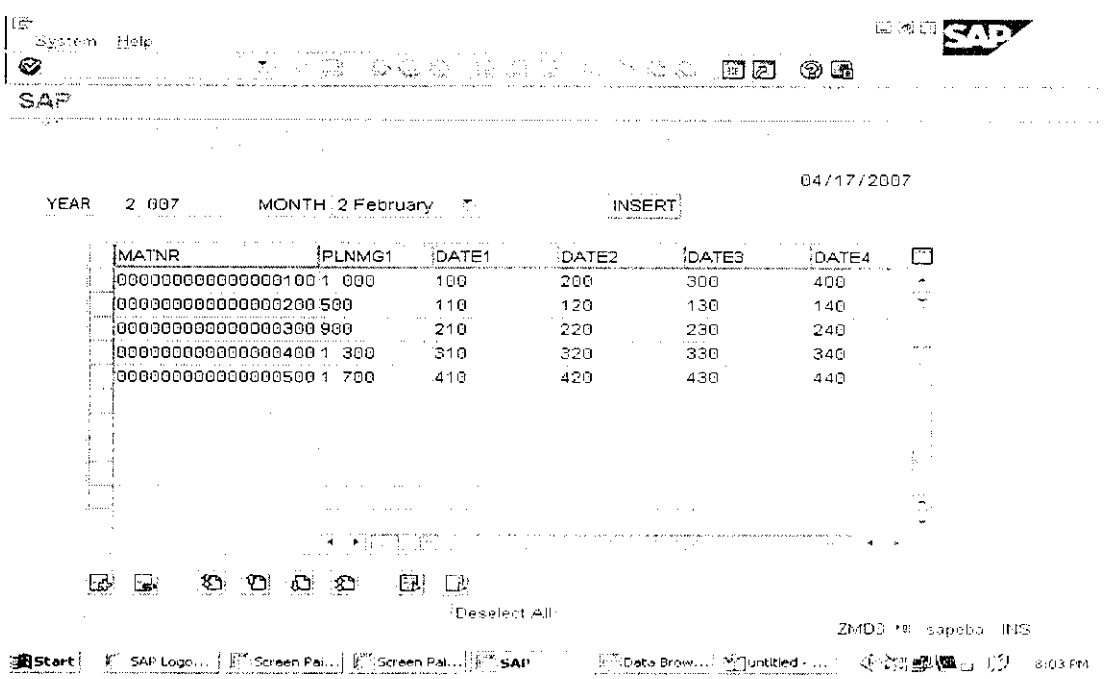


FIGURE 4.16: TRANSACTION SCREEN- showing deselecting all rows

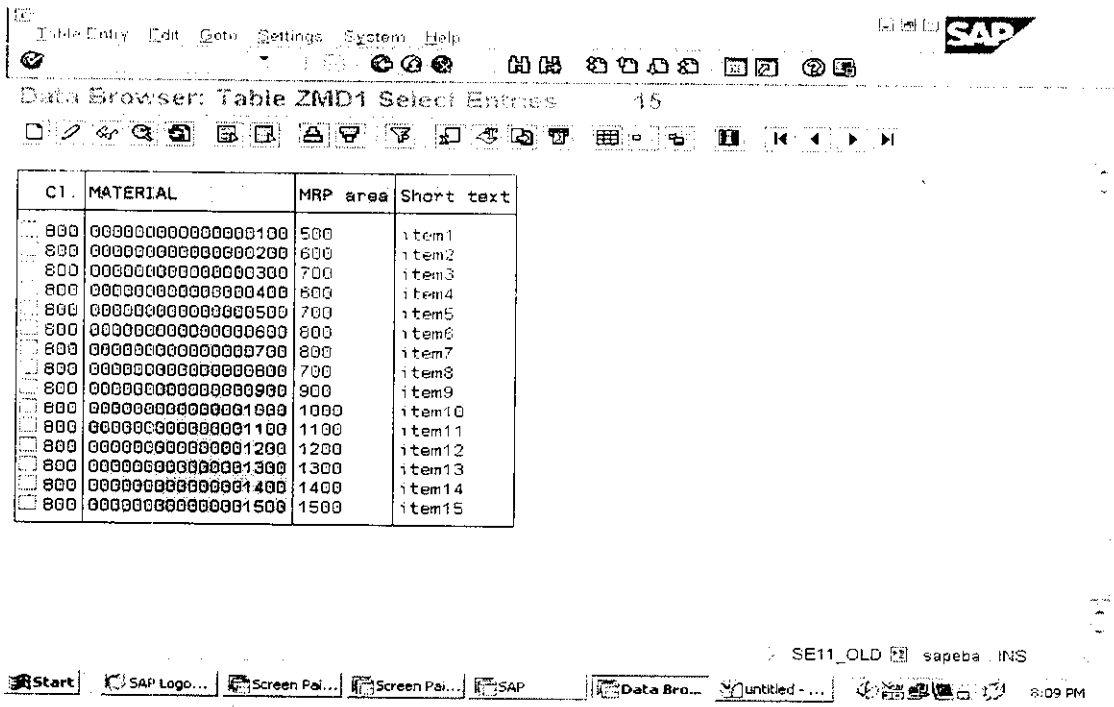


FIGURE 4.17: database table-Material Details

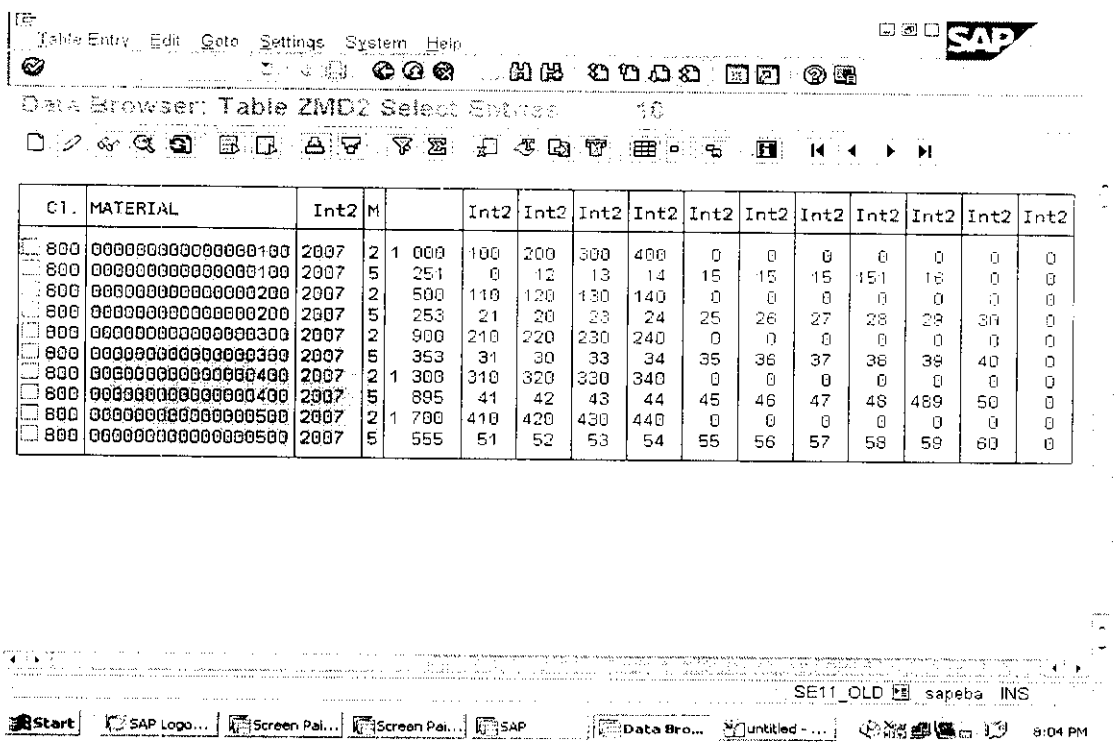


FIGURE 4.18: database table-for storing demand from PIR

Chapter 5

Conclusion

CONCLUSION

With the help of Finite Capacity model the company can obtain the following benefits. The maximum utilization of resources, no delay in making product and the products are to be delivered at right time. So many organizations trying to follow Finite Capacity model.

Finite Capacity model with Planned Independent Requirements in SAP will reduce the pressure of Production department. The company no need to invest money to buy a new module called Advanced Planning Optimization. This Module Specifically designed for detailed Planning and Scheduling.

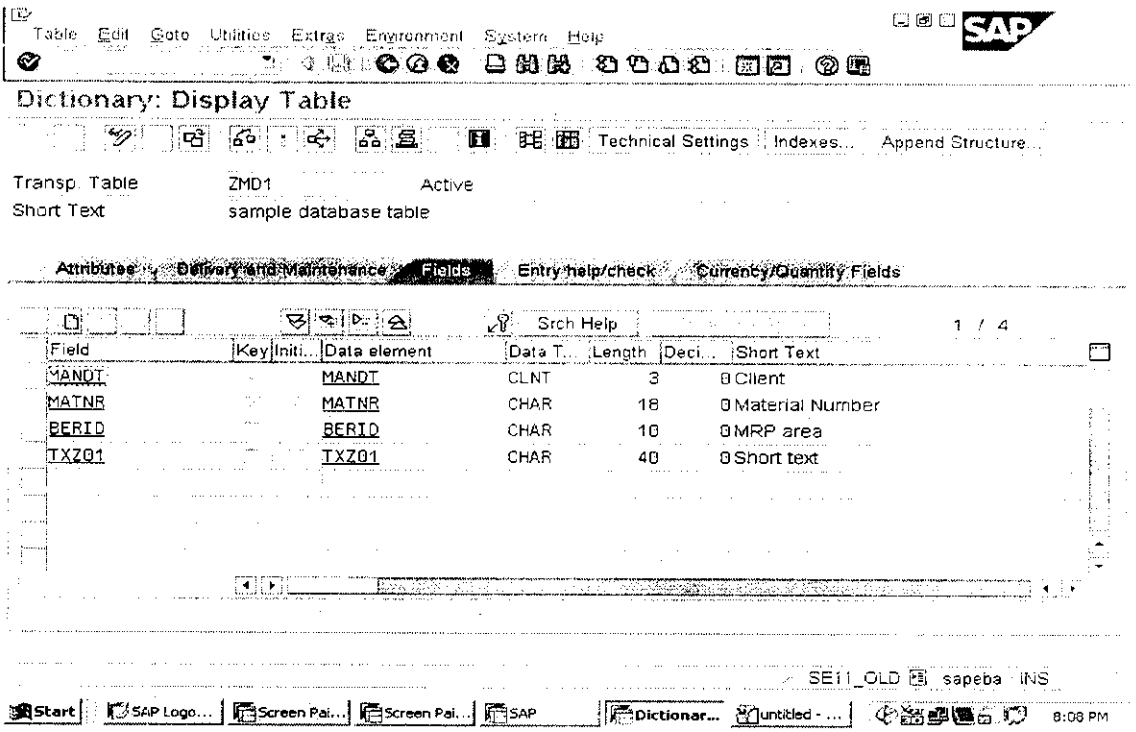
The major contribution of this research work is to prepare Transaction Screen for Planned Independent Requirements with Finite Capacity model, to control the demand when it exceeds beyond the finite capacity level. So based on this model, company can effectively prepare accurate budget.

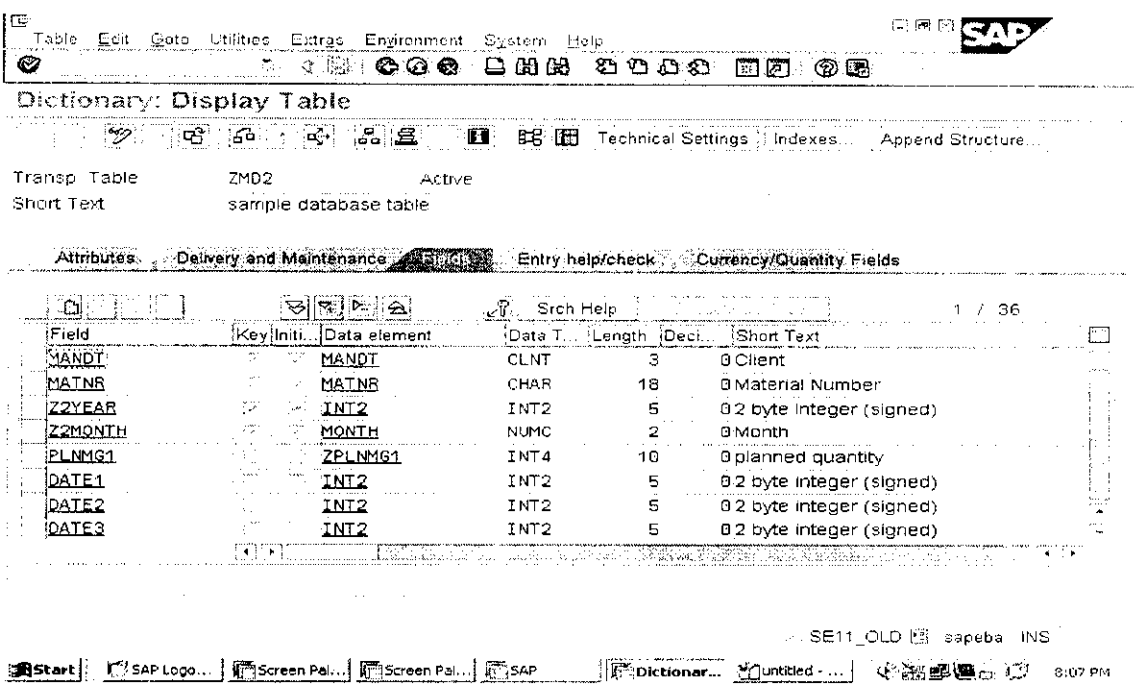
This research work has been successfully implemented in ROOTS industries, Coimbatore. To obtain the maximum Utilization of resources the demand of customer and resources should be balanced each other. The company has implemented SAP package in their organization. The existing system follows infinite capacity for demand. This research work will be very useful for them to follow finite capacity model. So that they can avoid bottleneck problem in the production department at starting stage itself. By implementing the newly developed finite capacity model the company will achieve maximum utilization of resources.

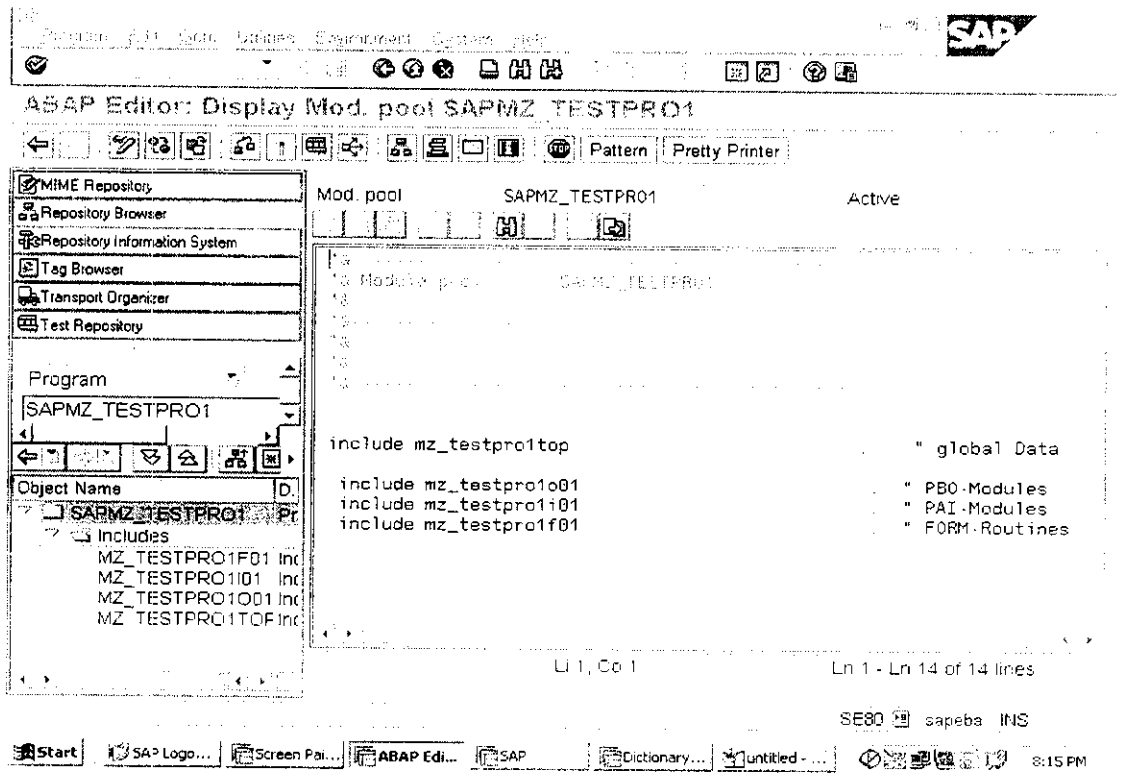
Appendices

APPENDIX 1

DATABASE TABLE and Developing Screen






ABAP Editor: Display Mod. pool SAPMZ_TESTPRO1
Pattern
Pretty Printer
MIME Repository
Repository Browser
Repository Information System
Tag Browser
Transport Organizer
Test Repository
Program
SAPMZ_TESTPRO1
Object Name
SAPMZ_TESTPRO1
Includes
MZ_TESTPRO1F01
MZ_TESTPRO1I01
MZ_TESTPRO1O01
MZ_TESTPRO1TOP
Mod. pool
SAPMZ_TESTPRO1
Active
include mz_testpro1top " global Data
include mz_testpro1o01 " PBO-Modules
include mz_testpro1i01 " PAI-Modules
include mz_testpro1f01 " FORM-Routines
Ln 1, Co 1
Ln 1 - Ln 14 of 14 lines
SE80
sapeba
INS
Start
ABAP Edi...
SAP
Dictionary...
8:15 PM

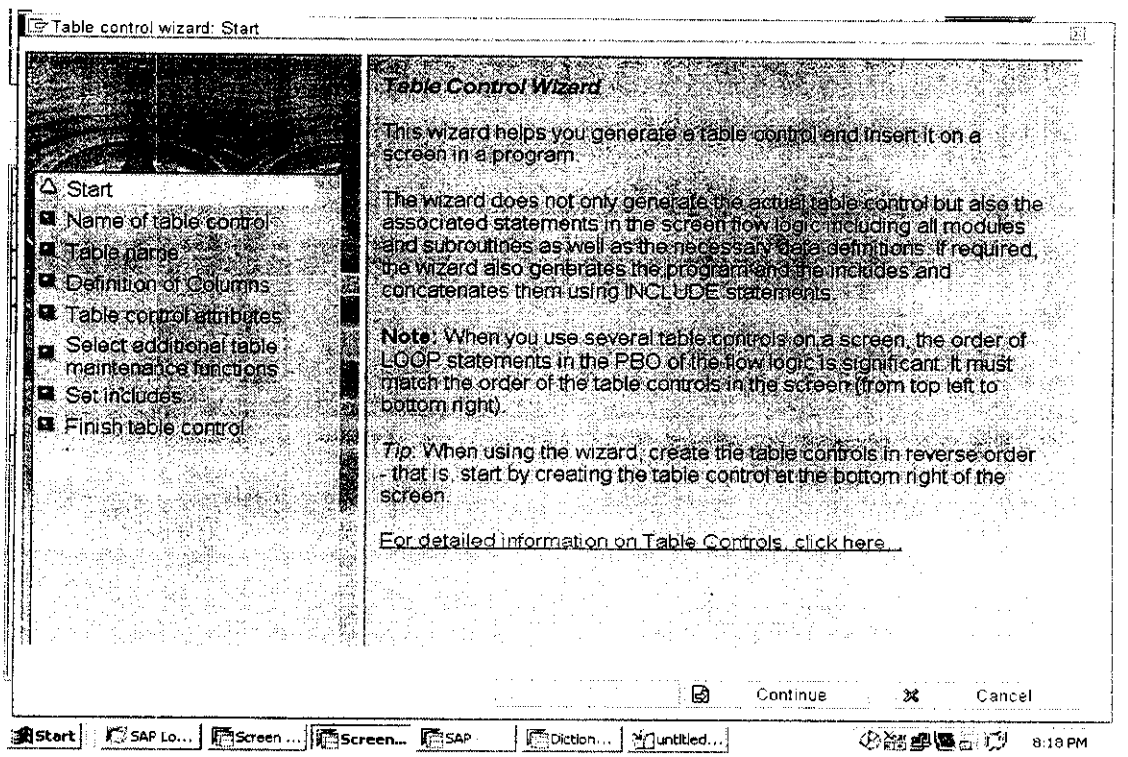


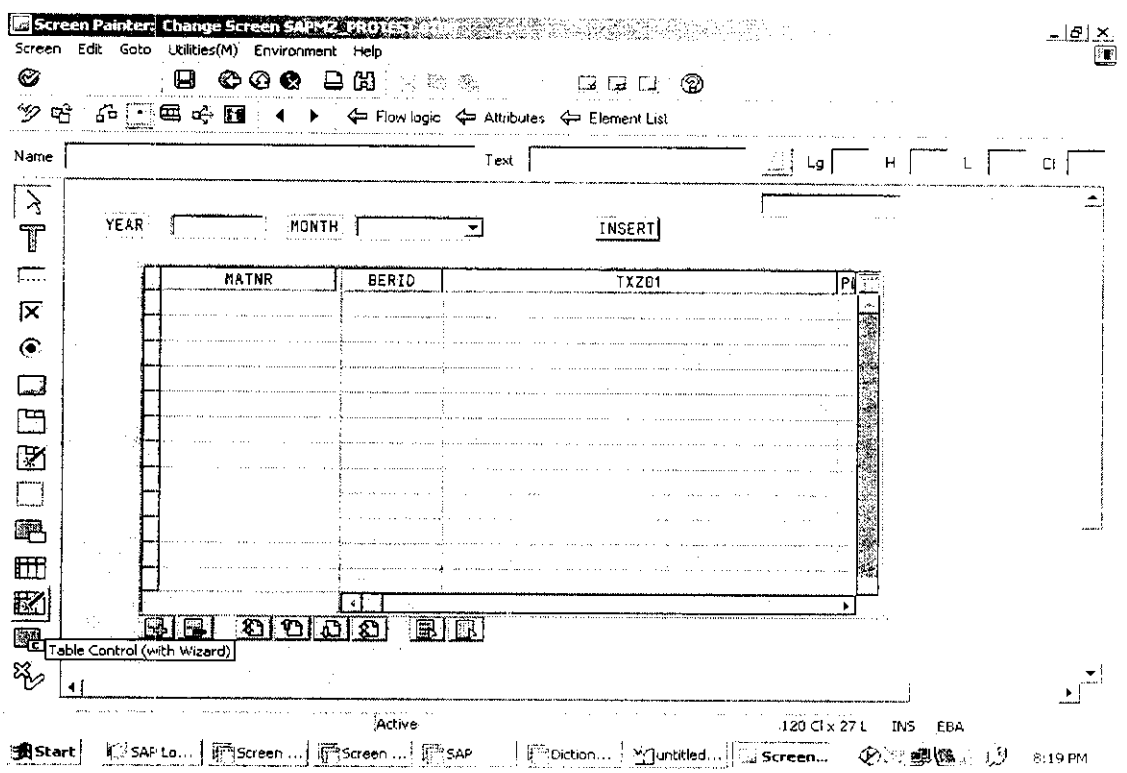
Table control wizard: Start
Start
Name of table control
Table name
Definition of Columns
Table control attributes
Select additional table maintenance functions
Set includes
Finish table control
Table Control Wizard
This wizard helps you generate a table control and insert it on a screen in a program.
The wizard does not only generate the actual table control but also the associated statements in the screen flow logic including all modules and subroutines as well as the necessary data definitions. If required, the wizard also generates the program and the includes and concatenates them using INCLUDE statements.
Note: When you use several table controls on a screen, the order of LOOP statements in the PBO of the flow logic is significant. It must match the order of the table controls in the screen (from top left to bottom right).
Tip: When using the wizard, create the table controls in reverse order - that is, start by creating the table control at the bottom right of the screen.
For detailed information on Table Controls, click here...
Continue
Cancel
Start
SAP Lo...
Screen ...
Screen...
SAP
Diction...
untitled...
8:18 PM

To create the screen two methods can be followed.

- Table control with Wizard
- Table control without wizard.

For working with second method user have to complete eight steps to develop the screen. The steps are,

- Start
- Name of the table control
- Table name
- Definition of column
- Table control attributes
- Select additional table maintenance functions.
- Set includes
- Finish table control.



APPENDIX 2

CODING

```
*&---------------------------------------------------------------------*
*& Include        MZ_PROTESTF01
*&---------------------------------------------------------------------*

*---------------------------------------------------------------------*
*  INCLUDE TABLECONTROL_FORMS                                         *
*---------------------------------------------------------------------*

*&---------------------------------------------------------------------*
*&      Form  USER_OK_TC                                              *
*&---------------------------------------------------------------------*
 FORM USER_OK_TC USING    P_TC_NAME TYPE DYNFNAM
                          P_TABLE_NAME
                          P_MARK_NAME
                 CHANGING P_OK      LIKE SY-UCOMM.

*&SPWIZARD: BEGIN OF LOCAL DATA----------------------------------------*
  DATA: L_OK            TYPE SY-UCOMM,
        L_OFFSET        TYPE I.
*&SPWIZARD: END OF LOCAL DATA------------------------------------------*

*&SPWIZARD: Table control specific operations
*&SPWIZARD: evaluate TC name and operations
  SEARCH P_OK FOR P_TC_NAME.
  IF SY-SUBRC <> 0.
    EXIT.
  ENDIF.
  L_OFFSET = STRLEN( P_TC_NAME ) + 1.
  L_OK = P_OK+L_OFFSET.
*&SPWIZARD: execute general and TC specific operations
  CASE L_OK.
    WHEN 'INSR'.                "insert row
      PERFORM FCODE_INSERT_ROW USING    P_TC_NAME
                                        P_TABLE_NAME.
      CLEAR P_OK.

    WHEN 'DELE'.                "delete row
      PERFORM FCODE_DELETE_ROW USING    P_TC_NAME
                                        P_TABLE_NAME
                                        P_MARK_NAME.
      CLEAR P_OK.

    WHEN 'P--' OR               "top of list
         'P-'  OR               "previous page
         'P+'  OR               "next page
         'P++'.                 "bottom of list
      PERFORM COMPUTE_SCROLLING_IN_TC USING P_TC_NAME
                                            L_OK.
      CLEAR P_OK.
*     WHEN 'L--'.               "total left
*       PERFORM FCODE_TOTAL_LEFT USING P_TC_NAME.
*
```

KCT LIBRARY Coimbatore - 641 006
P-2068

```
*    WHEN 'L-'.                  "column left
*     PERFORM FCODE_COLUMN_LEFT USING P_TC_NAME.
*
*    WHEN 'R+'.                  "column right
*     PERFORM FCODE_COLUMN_RIGHT USING P_TC_NAME.
*
*    WHEN 'R++'.                 "total right
*     PERFORM FCODE_TOTAL_RIGHT USING P_TC_NAME.
*
    WHEN 'MARK'.                 "mark all filled lines
     PERFORM FCODE_TC_MARK_LINES USING P_TC_NAME
                                 P_TABLE_NAME
                                 P_MARK_NAME .
     CLEAR P_OK.

    WHEN 'DMRK'.                 "demark all filled lines
     PERFORM FCODE_TC_DEMARK_LINES USING P_TC_NAME
                                  P_TABLE_NAME
                                  P_MARK_NAME .
     CLEAR P_OK.

*    WHEN 'SASCEND'  OR
*      'SDESCEND'.               "sort column
*     PERFORM FCODE_SORT_TC USING P_TC_NAME
*                          l_ok.

  ENDCASE.

 ENDFORM.                        " USER_OK_TC

*&---------------------------------------------------------------------*
*&     Form  FCODE_INSERT_ROW                                          *
*&---------------------------------------------------------------------*
 FORM fcode_insert_row
          USING   P_TC_NAME        TYPE DYNFNAM
                  P_TABLE_NAME          .

*&SPWIZARD: BEGIN OF LOCAL DATA----------------------------------------*
  DATA L_LINES_NAME     LIKE FELD-NAME.
  DATA L_SELLINE        LIKE SY-STEPL.
  DATA L_LASTLINE       TYPE I.
  DATA L_LINE           TYPE I.
  DATA L_TABLE_NAME     LIKE FELD-NAME.
  FIELD-SYMBOLS <TC>            TYPE CXTAB_CONTROL.
  FIELD-SYMBOLS <TABLE>         TYPE STANDARD TABLE.
  FIELD-SYMBOLS <LINES>         TYPE I.
*&SPWIZARD: END OF LOCAL DATA------------------------------------------*

  ASSIGN (P_TC_NAME) TO <TC>.

*&SPWIZARD: get the table, which belongs to the tc    *
  CONCATENATE P_TABLE_NAME '[]' INTO L_TABLE_NAME. "table body
  ASSIGN (L_TABLE_NAME) TO <TABLE>.             "not headerline

*&SPWIZARD: get looplines of TableControl *
  CONCATENATE 'G_' P_TC_NAME '_LINES' INTO L_LINES_NAME.
```

```
  ASSIGN (L_LINES_NAME) TO <LINES>.

*&SPWIZARD: get current line  *
  GET CURSOR LINE L_SELLINE.
  IF SY-SUBRC <> 0.                " append line to table
    L_SELLINE = <TC>-LINES + 1.
*&SPWIZARD: set top line *
    IF L_SELLINE > <LINES>.
      <TC>-TOP_LINE = L_SELLINE - <LINES> + 1.
    ELSE.
      <TC>-TOP_LINE = 1.
    ENDIF.
  ELSE.                            " insert line into table
    L_SELLINE = <TC>-TOP_LINE + L_SELLINE - 1.
    L_LASTLINE = <TC>-TOP_LINE + <LINES> - 1.
  ENDIF.
*&SPWIZARD: set new cursor line*
  L_LINE = L_SELLINE - <TC>-TOP_LINE + 1.

*&SPWIZARD: insert initial line*
  INSERT INITIAL LINE INTO <TABLE> INDEX L_SELLINE.
  <TC>-LINES = <TC>-LINES + 1.
*&SPWIZARD: set cursor*
  SET CURSOR LINE L_LINE.

 ENDFORM.                          " FCODE_INSERT_ROW

*&---------------------------------------------------------------------*
*&      Form  FCODE_DELETE_ROW                                         *
*&---------------------------------------------------------------------*
 FORM fcode_delete_row
          USING    P_TC_NAME          TYPE DYNFNAM
                   P_TABLE_NAME
                   P_MARK_NAME   .

*&SPWIZARD: BEGIN OF LOCAL DATA----------------------------------------*
  DATA L_TABLE_NAME      LIKE FELD-NAME.

  FIELD-SYMBOLS <TC>         TYPE cxtab_control.
  FIELD-SYMBOLS <TABLE>      TYPE STANDARD TABLE.
  FIELD-SYMBOLS <WA>.
  FIELD-SYMBOLS <MARK_FIELD>.
*&SPWIZARD: END OF LOCAL DATA------------------------------------------*

  ASSIGN (P_TC_NAME) TO <TC>.

*&SPWIZARD: get the table, which belongs to the tc*
  CONCATENATE P_TABLE_NAME '[]' INTO L_TABLE_NAME. "table body
  ASSIGN (L_TABLE_NAME) TO <TABLE>.              "not headerline

*&SPWIZARD: delete marked lines *
  DESCRIBE TABLE <TABLE> LINES <TC>-LINES.

  LOOP AT <TABLE> ASSIGNING <WA>.

*&SPWIZARD: access to the component 'FLAG' of the table header*
```

```
    ASSIGN COMPONENT P_MARK_NAME OF STRUCTURE <WA> TO <MARK_FIELD>.

    IF <MARK_FIELD> = 'X'.
     DELETE <TABLE> INDEX SYST-TABIX.
     IF SY-SUBRC = 0.
      <TC>-LINES = <TC>-LINES - 1.
     ENDIF.
    ENDIF.
   ENDLOOP.

 ENDFORM.                    " FCODE_DELETE_ROW

*&---------------------------------------------------------------------*
*&      Form  COMPUTE_SCROLLING_IN_TC
*&---------------------------------------------------------------------*
*       text
*----------------------------------------------------------------------*
*      -->P_TC_NAME  name of tablecontrol
*      -->P_OK       ok code
*----------------------------------------------------------------------*
 FORM COMPUTE_SCROLLING_IN_TC USING    P_TC_NAME
                                       P_OK.
*&SPWIZARD: BEGIN OF LOCAL DATA----------------------------------------*
  DATA L_TC_NEW_TOP_LINE     TYPE I.
  DATA L_TC_NAME             LIKE FELD-NAME.
  DATA L_TC_LINES_NAME       LIKE FELD-NAME.
  DATA L_TC_FIELD_NAME       LIKE FELD-NAME.

  FIELD-SYMBOLS <TC>         TYPE cxtab_control.
  FIELD-SYMBOLS <LINES>      TYPE I.
*&SPWIZARD: END OF LOCAL DATA------------------------------------------*

  ASSIGN (P_TC_NAME) TO <TC>.
*&SPWIZARD: get looplines of TableControl *
  CONCATENATE 'G_' P_TC_NAME '_LINES' INTO L_TC_LINES_NAME.
  ASSIGN (L_TC_LINES_NAME) TO <LINES>.


*&SPWIZARD: is no line filled?*
  IF <TC>-LINES = 0.
*&SPWIZARD: yes, ... *
    L_TC_NEW_TOP_LINE = 1.
  ELSE.
*&SPWIZARD: no, ... *
    CALL FUNCTION 'SCROLLING_IN_TABLE'
      EXPORTING
        ENTRY_ACT         = <TC>-TOP_LINE
        ENTRY_FROM        = 1
        ENTRY_TO          = <TC>-LINES
        LAST_PAGE_FULL    = 'X'
        LOOPS             = <LINES>
        OK_CODE           = P_OK
        OVERLAPPING       = 'X'
      IMPORTING
        ENTRY_NEW         = L_TC_NEW_TOP_LINE
      EXCEPTIONS
```

```
*          NO_ENTRY_OR_PAGE_ACT = 01
*          NO_ENTRY_TO          = 02
*          NO_OK_CODE_OR_PAGE_GO = 03
          OTHERS               = 0.
  ENDIF.

*&SPWIZARD: get actual tc and column *
  GET CURSOR FIELD L_TC_FIELD_NAME
         AREA  L_TC_NAME.

  IF SYST-SUBRC = 0.
    IF L_TC_NAME = P_TC_NAME.
*&SPWIZARD: et actual column *
      SET CURSOR FIELD L_TC_FIELD_NAME LINE 1.
    ENDIF.
  ENDIF.

*&SPWIZARD: set the new top line*
  <TC>-TOP_LINE = L_TC_NEW_TOP_LINE.


 ENDFORM.                    " COMPUTE_SCROLLING_IN_TC

*&---------------------------------------------------------------------*
*&      Form  FCODE_TC_MARK_LINES
*&---------------------------------------------------------------------*
*       marks all TableControl lines
*----------------------------------------------------------------------*
*     -->P_TC_NAME  name of tablecontrol
*----------------------------------------------------------------------*
FORM FCODE_TC_MARK_LINES USING P_TC_NAME
                               P_TABLE_NAME
                               P_MARK_NAME.
*&SPWIZARD: EGIN OF LOCAL DATA----------------------------------------*
 DATA L_TABLE_NAME       LIKE FELD-NAME.

 FIELD-SYMBOLS <TC>        TYPE cxtab_control.
 FIELD-SYMBOLS <TABLE>     TYPE STANDARD TABLE.
 FIELD-SYMBOLS <WA>.
 FIELD-SYMBOLS <MARK_FIELD>.
*&SPWIZARD: END OF LOCAL DATA----------------------------------------*

 ASSIGN (P_TC_NAME) TO <TC>.

*&SPWIZARD: get the table, which belongs to the tc  *
  CONCATENATE P_TABLE_NAME '[]' INTO L_TABLE_NAME. "table body
  ASSIGN (L_TABLE_NAME) TO <TABLE>.               "not headerline

*&SPWIZARD: mark all filled lines*
 LOOP AT <TABLE> ASSIGNING <WA>.

*&SPWIZARD: access to the component 'FLAG' of the table header*
   ASSIGN COMPONENT P_MARK_NAME OF STRUCTURE <WA> TO <MARK_FIELD>.

   <MARK_FIELD> = 'X'.
 ENDLOOP.
```

```
ENDFORM.                              "fcode_tc_mark_lines

*&---------------------------------------------------------------------*
*&      Form  FCODE_TC_DEMARK_LINES
*&---------------------------------------------------------------------*
*       demarks all TableControl lines
*----------------------------------------------------------------------*
*     -- P_TC_NAME  name of tablecontrol
*----------------------------------------------------------------------*
FORM FCODE_TC_DEMARK_LINES USING P_TC_NAME
                         P_TABLE_NAME
                         P_MARK_NAME .
*&SPWIZARD: BEGIN OF LOCAL DATA----------------------------------------*
 DATA L_TABLE_NAME      LIKE FELD-NAME.

 FIELD-SYMBOLS <TC>       TYPE cxtab_control.
 FIELD-SYMBOLS <TABLE>     TYPE STANDARD TABLE.
 FIELD-SYMBOLS <WA>.
 FIELD-SYMBOLS <MARK_FIELD>.
*&SPWIZARD: END OF LOCAL DATA------------------------------------------*

 ASSIGN (P_TC_NAME) TO <TC>.

*&SPWIZARD: get the table, which belongs to the tc *
  CONCATENATE P_TABLE_NAME '[]' INTO L_TABLE_NAME. "table body
  ASSIGN (L_TABLE_NAME) TO <TABLE>.             "not headerline

*&SPWIZARD: demark all filled lines*
 LOOP AT <TABLE> ASSIGNING <WA>.

*&SPWIZARD: access to the component 'FLAG' of the table header *
   ASSIGN COMPONENT P_MARK_NAME OF STRUCTURE <WA> TO <MARK_FIELD>.

   <MARK_FIELD> = SPACE.
 ENDLOOP.
ENDFORM.                              "fcode_tc_mark_lines

*&---------------------------------------------------------------------*
*& Include MZ_PROTESTTOP                          Module pool     SAPMZ_PROTEST
*&
*&---------------------------------------------------------------------*

PROGRAM  SAPMZ_PROTEST.

tables: zmd1,zmd2.

data: year type i,
    month(2),
    x type i,
    y type i.

data: begin of itab occurs 0,
matnr like zmd1-matnr,
berid like zmd1-berid,
txz01 like zmd1-txz01,
plnmg1 like zmd2-plnmg1,
```

```
date1 type i,date2 type i,date3 type i,date4 type i,
date5 type i,date6 type i,date7 type i,date8 type i,
date9 type i,date10 type i,date11 type i,date12 type i,
date13 type i,date14 type i,date15 type i,date16 type i,
date17 type i,date18 type i,date19 type i,date20 type i,
date21 type i,date22 type i,date23 type i,date24 type i,
date25 type i,date26 type i,date27 type i,date28 type i,
date29 type i,date30 type i,date31 type i,
chk,
end of itab.

DATA: ITAB1 LIKE ZMD2 OCCURS 0 WITH HEADER LINE.
data: date like sy-ucomm.
data: dd type string,
      mm type string,
      yyyy type string.

*&SPWIZARD: DECLARATION OF TABLECONTROL 'TC1' ITSELF
CONTROLS: TC1 TYPE TABLEVIEW USING SCREEN 0100.

*&SPWIZARD: LINES OF TABLECONTROL 'TC1'
DATA:    G_TC1_LINES  LIKE SY-LOOPC.

DATA:    OK_CODE LIKE SY-UCOMM.

*********DY-pro**********************
process before output.
*&SPWIZARD: PBO FLOW LOGIC FOR TABLECONTROL 'TC1'
 module tc1_change_tc_attr.
*&SPWIZARD: MODULE TC1_CHANGE_COL_ATTR.
 loop at  itab
     with control tc1
     cursor tc1-current_line.
   module tc1_get_lines.
*&SPWIZARD:   MODULE TC1_CHANGE_FIELD_ATTR
 endloop.

module status_0100.
module date.
module disable_date.

process after input.
*&SPWIZARD: PAI FLOW LOGIC FOR TABLECONTROL 'TC1'
 loop at itab.
  chain.
    field itab-matnr.
    field itab-berid.
    field itab-txz01.
    field itab-pinmg1.
    field itab-date1.
    field itab-date2.
    field itab-date3.
    field itab-date4.
    field itab-date5.
    field itab-date6.
    field itab-date7.
```

```
    field itab-date8.
    field itab-date9.
    field itab-date10.
    field itab-date11.
    field itab-date12.
    field itab-date13.
    field itab-date14.
    field itab-date15.
    field itab-date16.
    field itab-date17.
    field itab-date18.
    field itab-date19.
    field itab-date20.
    field itab-date21.
    field itab-date22.
    field itab-date23.
    field itab-date24.
    field itab-date25.
    field itab-date26.
    field itab-date27.
    field itab-date28.
    field itab-date29.
    field itab-date30.
    field itab-date31.
    module tc1_modify on chain-input.
    module denad_checking.
  endchain.
  MODULE ADDING_DEMAND.
  MODULE DATABASE_UPLOAD.
  field itab-chk
   module tc1_mark on request.
 endloop.
 module tc1_user_command.
*&SPWIZARD: MODULE TC1_CHANGE_TC_ATTR.
*&SPWIZARD: MODULE TC1_CHANGE_COL_ATTR.

MODULE USER_COMMAND_0100.
module year.
 field month module mon_change on request.
 field year module year_change on request.

*&---------------------------------------------------------------------*
*&  Include         MZ_PROTESTO01
*&---------------------------------------------------------------------*

*&SPWIZARD: OUTPUT MODULE FOR TC 'TC1'. DO NOT CHANGE THIS LINE!
*&SPWIZARD: UPDATE LINES FOR EQUIVALENT SCROLLBAR
MODULE TC1_CHANGE_TC_ATTR OUTPUT.
 DESCRIBE TABLE ITAB LINES TC1-lines.
ENDMODULE.

*&SPWIZARD: OUTPUT MODULE FOR TC 'TC1'. DO NOT CHANGE THIS LINE!
*&SPWIZARD: GET LINES OF TABLECONTROL
MODULE TC1_GET_LINES OUTPUT.
 G_TC1_LINES = SY-LOOPC.
ENDMODULE.
```

```
*&---------------------------------------------------------------------*
*&      Module  DATE  OUTPUT
*&---------------------------------------------------------------------*
*       text
*----------------------------------------------------------------------*
module DATE output.
DATE = SY-DATUM.
YYYY = DATE(4).
endmodule.                 " DATE  OUTPUT
*&---------------------------------------------------------------------*
*&      Module  DISABLE_DATE  OUTPUT
*&---------------------------------------------------------------------*
*       text
*----------------------------------------------------------------------*
module DISABLE_DATE output.
x = year mod 4.
 if x = 0.
 if month = 02.
 loop at screen.
 if screen-group3 = 'G3'.
 screen-INVISIBLE = '1'.
 SCREEN-INPUT = '0'.
 SCREEN-ACTIVE = '1'.
 modify screen.
 endif.
 endloop.
 endif.

 if month  = 04.
 loop at screen.
 if screen-group2 = 'G2'.
 screen-active = 0.
 modify screen.
 endif.
 endloop.
 endif.

  IF MONTH = 06.
loop at screen.
IF SCREEN-group2 = 'G2'.
*SCREEN-ACTIVE = '1'.
SCREEN-INPUT = '0'.
*SCREEN-OUTPUT = '1'.
*screen-INVISIBLE = '1'.
modify screen.
endif.
endloop.
ENDIF.

IF MONTH = 09.
loop at screen.
IF SCREEN-group2 = 'G2'.
*SCREEN-ACTIVE = '1'.
SCREEN-INPUT = '0'.
*SCREEN-OUTPUT = '1'.
*screen-INVISIBLE = '1'.
```

```
modify screen.
endif.
endloop.
ENDIF.

IF MONTH = 11.
loop at screen.
IF SCREEN-group2 = 'G2'.
SCREEN-ACTIVE = '1'.
SCREEN-INPUT = '0'.
SCREEN-OUTPUT = '1'.
screen-INVISIBLE = '1'.
modify screen.
endif.
endloop.
ENDIF.

elseif x > 0.
 if month = 02.
loop at screen.
IF SCREEN-group1 = 'G1'.
SCREEN-ACTIVE = 0.
*SCREEN-INPUT = '0'.
*SCREEN-OUTPUT = '1'.
*screen-INVISIBLE = '1'.
modify screen.
endif.
endloop.
ENDIF.

IF MONTH = 04.
loop at screen.
IF SCREEN-group2 = 'G2'.
SCREEN-ACTIVE = '1'.
SCREEN-INPUT = '0'.
SCREEN-OUTPUT = '1'.
screen-INVISIBLE = '1'.
modify screen.
endif.
endloop.
ENDIF.

IF MONTH = 06.
loop at screen.
IF SCREEN-group2 = 'G2'.
SCREEN-ACTIVE = '1'.
SCREEN-INPUT = '0'.
SCREEN-OUTPUT = '1'.
screen-INVISIBLE = '1'.
modify screen.
endif.
endloop.
ENDIF.

IF MONTH = 09.
loop at screen.
```

```
IF SCREEN-group2 = 'G2'.
SCREEN-ACTIVE = '1'.
SCREEN-INPUT = '0'.
SCREEN-OUTPUT = '1'.
screen-INVISIBLE = '1'.
modify screen.
endif.
endloop.
ENDIF.

IF MONTH = 11.
loop at screen.
IF SCREEN-group2 = 'G2'.
SCREEN-ACTIVE = '1'.
SCREEN-INPUT = '0'.
SCREEN-OUTPUT = '1'.
screen-INVISIBLE = '1'.
modify screen.
endif.
endloop.
ENDIF.
endif.
endmodule.            " DISABLE_DATE  OUTPUT

*&---------------------------------------------------------------------*
*&  Include         MZ_PROTESTI01
*&---------------------------------------------------------------------*

*&SPWIZARD: INPUT MODULE FOR TC 'TC1'. DO NOT CHANGE THIS LINE!
*&SPWIZARD: MODIFY TABLE
MODULE TC1_MODIFY INPUT.
 MODIFY ITAB
  INDEX TC1-CURRENT_LINE.
  READ TABLE ITAB WITH KEY MATNR = ITAB-MATNR.
  IF SY-SUBRC <> 0.
    APPEND ITAB.
    ELSE.
    MODIFY ITAB INDEX TC1-CURRENT_LINE.
  ENDIF.
 CALL FUNCTION 'CONVERSION_EXIT_ALPHA_INPUT'
 EXPORTING
  input      = itab-matnr
IMPORTING
 OUTPUT      = itab-matnr.
 SELECT SINGLE * FROM ZPro1 INTO CORRESPONDING FIELDS OF
       ITAB WHERE MATNR = ITAB-MATNR.
 if itab-date1 is initial and
  itab-date2 is initial and
  itab-date3 is initial .
 select single * from zMD2 into corresponding fields of zMD2
       where z2year = year and z2month = month
       and matnr = itab-matnr.
else.
move-corresponding itab to zMD2.
endif.
if sy-subrc = 0.
```

```
 move-corresponding zMD2 to itab.
endif.
modify itab index tc1-current_line transporting
                          matnr
                          txz01
                          berid
                         *plnmg1
                          date1 date2 date3 date4 date5
                          date6 date7 date8 date9 date10
                          date11 date12 date13 date14
                          date15 date16 date17 date18
                          date19 date20 date21 date22
                          date23 date24 date25 date26
                          date27 date28 date29 date30
                          date31.
 if sy-subrc <> 0.
      insert itab index tc1-current_line.
   endif.
ENDMODULE.

*&SPWIZARD: INPUT MODUL FOR TC 'TC1'. DO NOT CHANGE THIS LINE!
*&SPWIZARD: MARK TABLE
MODULE TC1_MARK INPUT.
 DATA: g_TC1_wa2 like line of ITAB.
  if TC1-line_sel_mode = 1
  and ITAB-CHK = 'X'.
   loop at ITAB into g_TC1_wa2
     where CHK = 'X'.
     g_TC1_wa2-CHK = ''.
     modify ITAB
      from g_TC1_wa2
      transporting CHK.
   endloop.
 endif.
 MODIFY ITAB
  INDEX TC1-CURRENT_LINE
  TRANSPORTING CHK.
ENDMODULE.

*&SPWIZARD: INPUT MODULE FOR TC 'TC1'. DO NOT CHANGE THIS LINE!
*&SPWIZARD: PROCESS USER COMMAND
MODULE TC1_USER_COMMAND INPUT.
 OK_CODE = SY-UCOMM.
 PERFORM USER_OK_TC USING    'TC1'
                     'ITAB'
                     'CHK'
              CHANGING OK_CODE.
 SY-UCOMM = OK_CODE.
ENDMODULE.
*&---------------------------------------------------------------------*
*&      Module  DENAD_CHECKING  INPUT
*&---------------------------------------------------------------------*
*       text
*----------------------------------------------------------------------*
module DENAD_CHECKING input.
if itab-date1 > itab-berid. "or itab-date2 > itab-berid.
```

```
clear itab-date1.
message 'DATE1 DEMAND IS OVERLODE' type 'E'.
else.
modify itab index tc1-current_line transporting date1.
  if sy-subrc <> 0.
    insert itab index tc1-current_line.
  endif.
*clear itab-date1.
endif.
*ENDIF.

if itab-date2 > itab-berid.
clear itab-date2.
message 'DATE2 DEMAND IS OVERLOAD' type 'E'.
else.
modify itab index tc1-current_line transporting date2.
endif.

if itab-date3 > itab-berid.
clear itab-date3.
message 'DATE3 DEMAND IS OVERLODE' type 'E'.
else.
modify itab index tc1-current_line transporting date3.
endif.

if itab-date4 > itab-berid.
clear itab-date4.
message 'DATE4 DEMAND IS OVERLODE' type 'E'.
else.
modify itab index tc1-current_line transporting date4.
endif.

if itab-date5 > itab-berid.
clear itab-date5.
message 'DATE5 DEMAND IS OVERLODE' type 'E'.
else.
modify itab index tc1-current_line transporting date5.
endif.

if itab-date6 > itab-berid.
clear itab-date6.
message 'DATE6 DEMAND IS OVERLODE' type 'E'.
else.
modify itab index tc1-current_line transporting date6.
endif.

if itab-date7 > itab-berid.
clear itab-date7.
message 'DATE7 DEMAND IS OVERLODE' type 'E'.
else.
modify itab index tc1-current_line transporting date7.
endif.

if itab-date8 > itab-berid.
clear itab-date8.
message 'DATE8 DEMAND IS OVERLODE' type 'E'.
```

```
else.
modify itab index tc1-current_line transporting date8.
endif.

if itab-date9 > itab-berid.
clear itab-date9.
message 'DATE9 DEMAND IS OVERLODE' type 'E'.
else.
modify itab index tc1-current_line transporting date9.
endif.

if itab-date10 > itab-berid.
clear itab-date10.
message 'DATE10 DEMAND IS OVERLODE' type 'E'.
else.
modify itab index tc1-current_line transporting date10.
endif.

if itab-date11 > itab-berid.
clear itab-date11.
message 'DATE11 DEMAND IS OVERLODE' type 'E'.
else.
modify itab index tc1-current_line transporting date11.
endif.

if itab-date12 > itab-berid.
clear itab-date12.
message 'DATE12 DEMAND IS OVERLODE' type 'E'.
else.
modify itab index tc1-current_line transporting date12.
endif.

if itab-date13 > itab-berid.
clear itab-date13.
message 'DATE13 DEMAND IS OVERLODE' type 'E'.
else.
modify itab index tc1-current_line transporting date13.
endif.

if itab-date14 > itab-berid.
clear itab-date14.
message 'DATE14 DEMAND IS OVERLODE' type 'E'.
else.
modify itab index tc1-current_line transporting date14.
endif.

if itab-date15 > itab-berid.
clear itab-date15.
message 'DATE15 DEMAND IS OVERLODE' type 'E'.
else.
modify itab index tc1-current_line transporting date15.
endif.

if itab-date16 > itab-berid.
clear itab-date16.
message 'DATE16 DEMAND IS OVERLODE' type 'E'.
```

```
else.
modify itab index tc1-current_line transporting date16.
endif.

if itab-date17 > itab-berid.
clear itab-date17.
message 'DATE17 DEMAND IS OVERLODE' type 'E'.
else.
modify itab index tc1-current_line transporting date17.
endif.

if itab-date18 > itab-berid.
clear itab-date18.
message 'DATE18 DEMAND IS OVERLODE' type 'E'.
else.
modify itab index tc1-current_line transporting date18.
endif.

if itab-date19 > itab-berid.
clear itab-date19.
message 'DATE19 DEMAND IS OVERLODE' type 'E'.
else.
modify itab index tc1-current_line transporting date19.
endif.

if itab-date20 > itab-berid.
clear itab-date20.
message 'DATE20 DEMAND IS OVERLODE' type 'E'.
else.
modify itab index tc1-current_line transporting date20.
endif.

if itab-date21 > itab-berid.
clear itab-date21.
message 'DATE21 DEMAND IS OVERLODE' type 'E'.
else.
modify itab index tc1-current_line transporting date21.
endif.

if itab-date22 > itab-berid.
clear itab-date22.
message 'DATE22 DEMAND IS OVERLODE' type 'E'.
else.
modify itab index tc1-current_line transporting date22.
endif.

if itab-date23 > itab-berid.
clear itab-date23.
message 'DATE23 DEMAND IS OVERLODE' type 'E'.
else.
modify itab index tc1-current_line transporting date23.
endif.

if itab-date24 > itab-berid.
clear itab-date24.
message 'DATE24 DEMAND IS OVERLODE' type 'E'.
```

```
else.
modify itab index tc1-current_line transporting date24.
endif.

if itab-date25 > itab-berid.
clear itab-date25.
message 'DATE25 DEMAND IS OVERLODE' type 'E'.
else.
modify itab index tc1-current_line transporting date25.
endif.

if itab-date26 > itab-berid.
clear itab-date26.
message 'DATE26 DEMAND IS OVERLODE' type 'E'.
else.
modify itab index tc1-current_line transporting date26.
endif.

if itab-date27 > itab-berid.
clear itab-date27.
message 'DATE27 DEMAND IS OVERLODE' type 'E'.
else.
modify itab index tc1-current_line transporting date27.
endif.

if itab-date28 > itab-berid.
clear itab-date28.
message 'DATE28 DEMAND IS OVERLODE' type 'E'.
else.
modify itab index tc1-current_line transporting date28.
endif.

if itab-date29 > itab-berid.
clear itab-date29.
message 'DATE29 DEMAND IS OVERLODE' type 'E'.
else.
modify itab index tc1-current_line transporting date29.
endif.

if itab-date30 > itab-berid.
clear itab-date30.
message 'DATE30 DEMAND IS OVERLODE' type 'E'.
else.
modify itab index tc1-current_line transporting date30.
endif.

if itab-date31 > itab-berid.
clear itab-date31.
message 'DATE31 DEMAND IS OVERLODE' type 'E'.
else.
modify itab index tc1-current_line transporting date31.
endif.


endmodule.              " DENAD_CHECKING  INPUT
```

```
*&---------------------------------------------------------------------*
*&      Module  ADDING_DEMAND  INPUT
*&---------------------------------------------------------------------*
*     text
*---------------------------------------------------------------------*

module ADDING_DEMAND input.
itab-plnmg1 = itab-date1 + itab-date2 + itab-date3 + itab-date4
+ itab-date5 + itab-date6 + itab-date7 + itab-date8 + itab-date9
+ itab-date10 + itab-date11 + itab-date12 + itab-date13
+ itab-date14 + itab-date15 + itab-date16 + itab-date17
+ itab-date18 + itab-date19 + itab-date20 + itab-date21
+ itab-date22 + itab-date23 + itab-date24 + itab-date25
+ itab-date26 + itab-date27 + itab-date28 + itab-date30
+ itab-date31.
modify itab index tc1-current_line transporting plnmg1.
endmodule.              " ADDING_DEMAND  INPUT

*&---------------------------------------------------------------------*
*&      Module  DATABASE_UPLOAD  INPUT
*&---------------------------------------------------------------------*
*     text
*---------------------------------------------------------------------*

module DATABASE_UPLOAD input.
case sy-ucomm.
when 'INSE'.
y = year.
*modify itab index tc1-current_line.
move-corresponding itab to itab1.
itab1-Z2year = y.
itab1-Z2month = month.
modify zpro2 from itab1 .
  if sy-subrc = 0.
   message ' RECORDS INSERTED' type 'S'.
  endif.
if not itab1 is initial.
modify zMD2 from itab1 .
endif.
endcase.
endmodule.              " DATABASE_UPLOAD  INPUT

*&---------------------------------------------------------------------*
*&      Module  year  INPUT
*&---------------------------------------------------------------------*
*     text
*---------------------------------------------------------------------*

module year input.
if year < yyyy.
message ' CURRENT OR FUTURE DEMAND ONLY ACCEPTED' type 'I'.
endif.
endmodule.              " year  INPUT

*&---------------------------------------------------------------------*
*&      Module  mon_change  INPUT
```

```
*&---------------------------------------------------------------------*
*      text
*----------------------------------------------------------------------*

module mon_change input.
REFRESH: ITAB.
endmodule.                " mon_change  INPUT

*&---------------------------------------------------------------------*
*&     Module  year_change  INPUT
*&---------------------------------------------------------------------*
*      text
*----------------------------------------------------------------------*

module year_change input.
REFRESH: ITAB.
endmodule.                " year_change  INPUT
```

References

REFERENCES

1. Collette Rolland and Naveen Prakash (2000), Bridging the gap between organizational needs and ERP functionality, Journal of Requirement Engineering, Vol.5, pp.180-193.

2. Fawzy Soliman and Mohamed A.Youssef (1998) ,The role of SAP Software in business process Re-engineering, Intenational Journal of Operation & Production Management, Vol.18,No.9/18,pp.886-895.

3. Gareth M.DeBruyn, Robert Lyfareff (2000), *Introduction to ABAP/4 Programming for SAP*, Revised and Expanded edition, Mumbai.

4. Helmut Klaus, Michel Roseman and Guy G.Gable (2000), Journal of Information Systems Frontiers 2:2, pp.141-162.

5. Irma Becerra Fernandez, Kenneth E. Murphy and Joyce Elam (2005), Successfully implementing EPR: The IBM Personnel Systems group experience , International Journal of Internet and Enterprise Management, Vol.3, pp. 78-97.

6. Jorge Cordoso, Robert P. Bostrom and Amit Smit(2004), Workflow Management systems and ERP Systems: Differences, Commonalities and Applications, Journal of Information Systems and Management,Vol.5, pp.319-338.

7. Judy E.Scott and Iris Vessy (200),.Implementing Enterprise Resource Planning Systems: The Role of learning from Failure, Journal of Information Systems Frontiers 2:2, pp.213-232.

8. Keith Porter , David Little , John Kenworthy and Peter Jorvys (1996), Finite Capacity Scheduling Tools: Observations of installations offer some lessons, Journal of Integrated Manufacturing Systems,Vol.No.7/4, pp.34-38.

9. Majed Al-Mashari and Mohamad Zairi (2000),The effective Application of SAP R/3 : a proposed model of best practice, Journal of Logistics and management, Vol.13, No.3,pp.156-66.

10. Maya Daneva.Roel J.Wieringa (2006), A requirement Engineering framework for Cross Organizational ERP systems, Journal of Requirement Engineering, Vol.11, pp.194-204.

11. Pernille kroemmergaard and Jeremy rose(2002), Management competences for ERP journeys, Journal of Information Systems Frontiers 4:2,pp.199-211.

12. Vidyaranya b. Gargeya and Cydnee Brady (2005), Success and Failure Factors of SAP in ERP System Implementation, Journal of Business Process Management, Vol.11, No.5, pp.501-516.

13. www.emeraldinsight.com.

14. www.springerlink.com.

15. www.taylorandfrancis.com.

16. www.indescience.com.

17. www.sciencedirect.com.

18. www.sapgenie.com.

19. www.sapfans.com.

20. www.mysap.com.

21. www.sapimg.com.

22. www.digilander.libero.it/vietrim/index.html.

23. www.journalsonline.tandf.co.uk/app.home.

24. www.sapbrain.com.