PART 1

Introduction

Abstract:

In recent years computer automation has become of great importance in business and service sectors. Our project is a simple demonstration of automation of a departmental store management system using Oracle, world's most popular relational database management system.

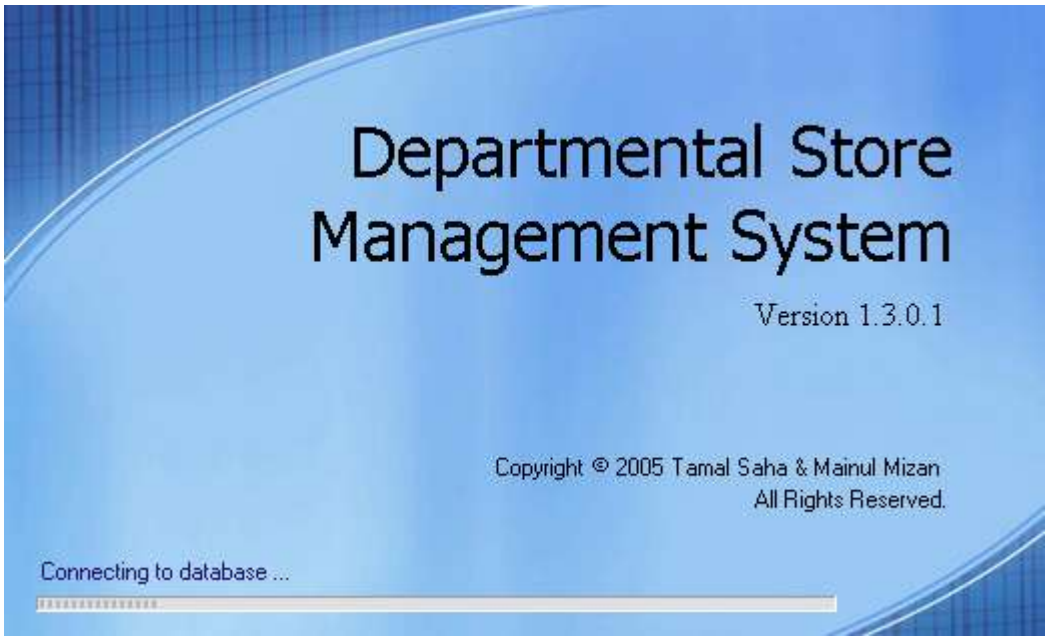

Figure #1: Departmental Store Management System Splash Screen

System Requirements:

Hardware Requirements

Table 1 lists the minimum hardware requirements:

Table 1: Hardware Requirements

Requirement	Minimum Value
Physical memory (RAM)	256 MB minimum, 512 MB recommended
Virtual memory	Double the amount of RAM
Temp disk space	100 MB
Hard disk space	2 GB
Video adapter	256 colors
Processor	200 MHz minimum

Software Requirements

Table 2 lists the software requirements for DSMS:

Table 2 Software Requirements

Requirement	Value
System Architecture	32-bit
Operating System	Departmental Store Management System (DSMS) for Windows is supported on the following operating systems: • Windows NT Server 4.0, Windows NT Server Enterprise Edition 4.0, and Terminal Server Edition with service pack 6a or higher are supported. Windows NT Workstation is no longer supported. • Windows 2000 with service pack 1 or higher. • Windows Server 2003 • Windows XP Professional

How to install:

Follow the three steps written below to install the Departmental Store Management System (DSMS) on your pc:

- Log In to the System with Administrator Privileges.
- Install .Net Framework 1.1 on your computer.
- Install Oracle 10g on your computer. For more detailed information about installing Oracle Database components, see one of the following guides:
  - Oracle® Database Quick Installation Guide 10*g* for Windows
  - *Oracle Database Installation Guide for Windows.*
- Open the DSMS.sql file in Oracle SQL*Plus Worksheet and press F5 to built the tables.
- Double click the DSMS.exe file and log in one of the three privilege modes using the user_id and password given below:

Table 3: Initial User ID and Password

User ID	Password	Privilege
q	q	ADMIN
a	a	SELLER
Z	z	PURCHASER

Results of a Successful Installation:

After you successfully install Departmental Store Management System (DSMS):

- The database that you created and the default Oracle Net listener process are running on the system.
- Oracle Enterprise Manager Database Control and iSQL*Plus are running and can be accessed using a Web browser.
- A single-node version of the Oracle Cluster Synchronization Services (CSS) daemon is running and is configured to start automatically when your system boots.

PART 2

Software Description

Main Features:

Departmental Store Management System (DSMS) will be capable of managing daily sales, purchases, damages, expenditures etc.

It can store information of a product in depth in a very structured way.

Invoices are created and can be printed whenever the shop sells, buys, returns, or report damages.

The interface of the system will be user friendly and as simple as possible yet powerful requiring little training of sales manager.

The following paragraphs describe the various features of different parts of the software.

Software Users:

During the design phase of DSMS software we tried to identify the different categories of user that would be using this software. It was apparent that some guy who is a sales person need not to access to employee or product manufacturer's details. Similarly a purchaser need not worry about preparing an invoice. Such thoughts led us to classify the users in three categories – Administrator, Seller and Purchaser. We then had two options - firstly we could build one integrated interface including all features, then showing only the appropriate features to a user, Secondly we could build three separate interfaces for the three categories each implementing category specific features. The first one had some drawbacks such as clutter and lower level of security, so we chose to go on with the second option. We ended up having three interfaces for Administrators, Seller and Purchaser.

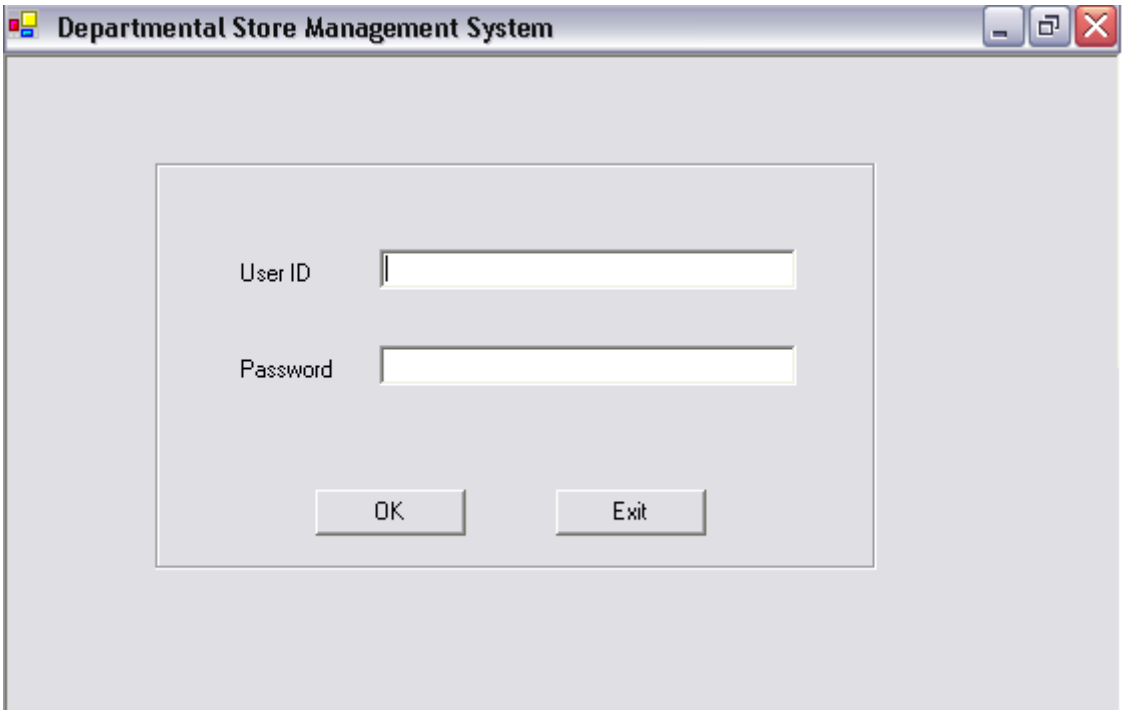

Figure #2: Departmental Store log on interface

The Administrator Interface:

We identified the following as the primary tasks of the administrator:

⇒ Add new product, edit product information and search product information.

⇒ Check the auto-generated orders, make new orders and explore all currently placed orders.

⇒ Add new supplier, edit supplier information and search supplier information.

⇒ Add new employee, edit employee information and search employee information.

⇒ Create new users with appropriate privilege, change user id and password, view all users both past and present and view user log on and log off time.

⇒ Add new expenses & damages and view all previous expenses and damages.

We implemented the above features in the administrator interface. The following is a screenshot from the administrator interface.

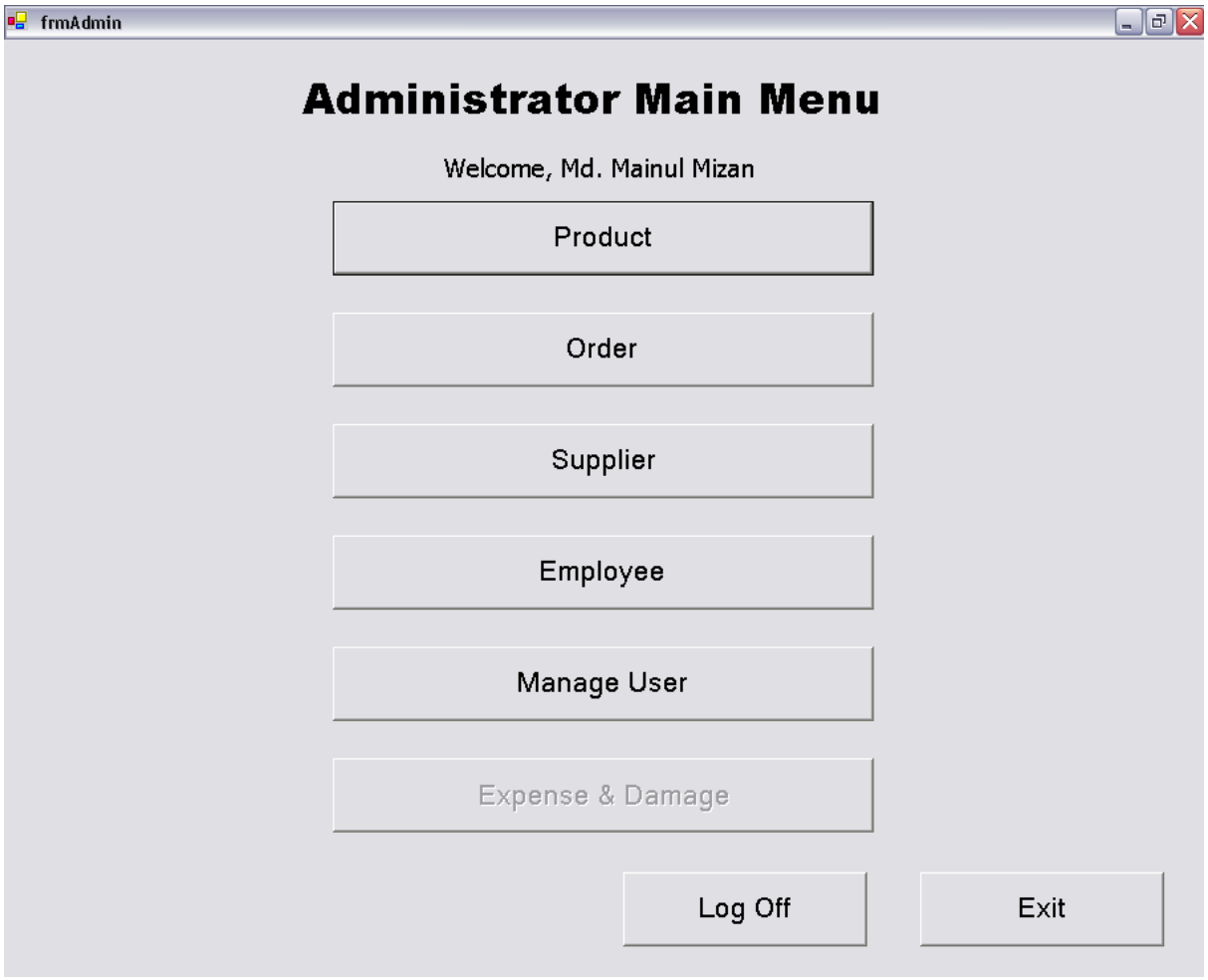

Figure #3: Departmental Store Administrator Interface

We summarize DSMS administrator's functionality below:

Product:

All the product related functionalities are grouped under the '**DSMS - Product Explorer**' form. Here the admin can add new products, modify the data such as product name, manufacturer name and address, minimum stock limit, unit description, product category and subcategory. The admin can also search product based on product id, product name, manufacturer name, product category and subcategory.

Order:

All the order related functionalities are grouped under the '**DSMS – Order Explorer**' form. Here the admin can see the auto generated orders by DSMS, create new orders and explore all the currently placed orders.

Supplier:

All the supplier related functionalities are grouped under the '**DSMS - Supplier Explorer**' form. Here the admin can add new suppliers, modify supplier data such as supplier id, supplier name & address, supplier ratings and add the new products to the currently selected supplier's supply list.

Employee:

All the employee related functionalities are grouped under the '**DSMS - Employee Explorer**' form. Here the admin can add new employees. Modification of employee information such as address, telephone number, salary, job title, work status, resignation date was not implemented due to time limitation but scope is there.

Manage User:

All the user related functionalities are grouped under the '**DSMS - User Explorer**' form. Here the admin can add new user and modify user information such as user id and password. The admin can also explore all the users of this software both present and past under the 'All Users' tab. The admin can also see the complete log on and log off time information under the 'view User Log' tab.

Damage and Expenses:

We did not implement damages and expenses functionality due to time limitation but scope is there for this functionality.

The Seller Interface:

We identified the following as the primary tasks of the seller:

⇒ Create new customer.

⇒ Prepare invoices.

⇒ Search previous invoices.

We implemented the above features in the seller interface. The following is a screenshot from the seller interface.

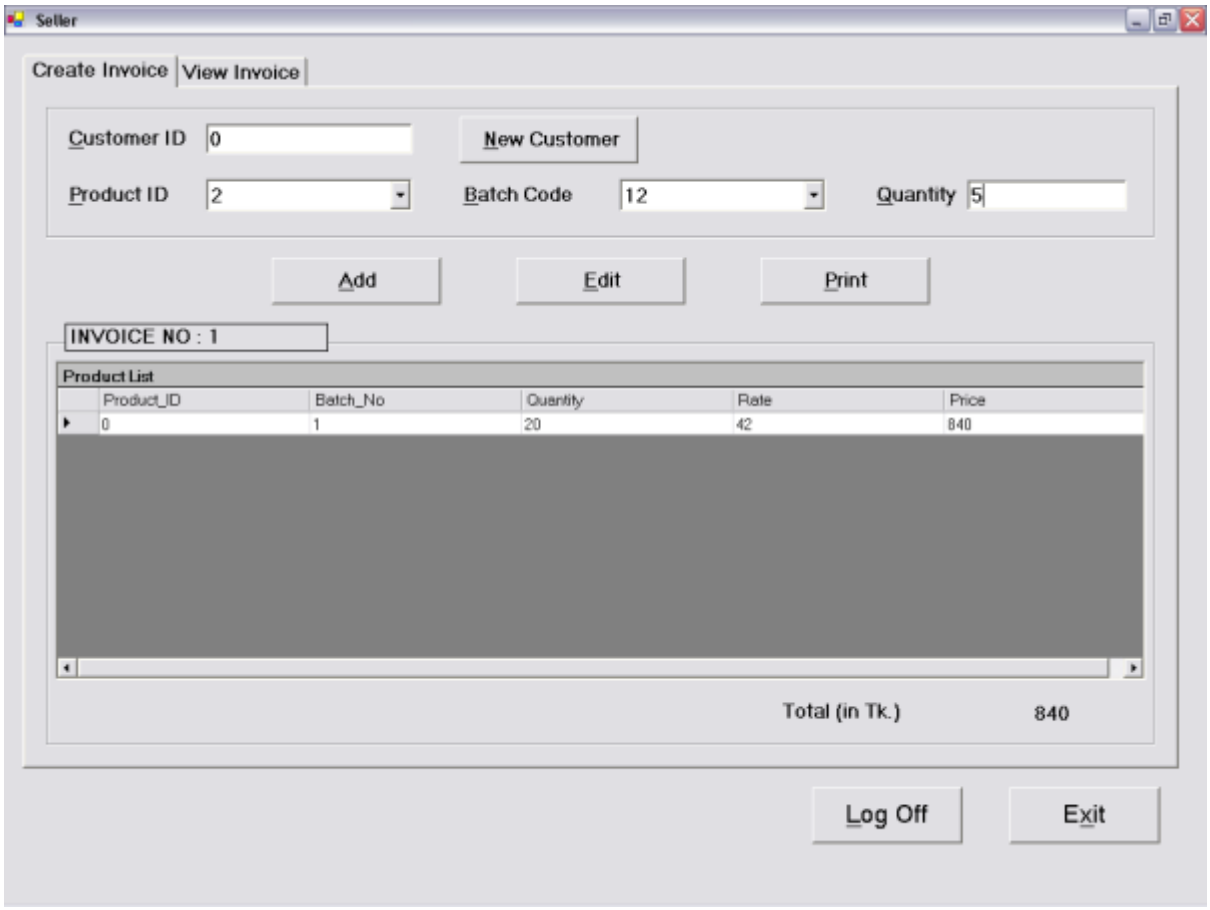

Figure #4: Departmental Store Seller Interface

We summarize DSMS seller's functionality below:

Customer:

All the customer related functionalities are grouped under the '**DSMS – New Customer**' form. Here the seller can add new customers. The seller can also select previous customer using their customer id from customer id dropdown list.

Create Invoice:

Functionalities related to creating new invoices are grouped under the '**Create Invoice**' tab. Here the seller can add new products in the invoice, change their amount. Currently the seller cannot print the invoice but scope is there to add this very important functionality.

Search Invoice:

All the functionalities related to searching an invoice are grouped under the '**View Invoice**' tab. Here the seller has to enter the invoice number and can view the customer id and the products sold under this invoice.

The Purchaser Interface:

We identified the following as the primary tasks of the purchaser:

⇒ Place the orders made available by the admin to the corresponding supplier.

⇒ Receive supply from the supplier.

⇒ Change the price of different lots of a product based on the current market price of that product.

We implemented the above features in the seller interface. The following is a screenshot from the purchaser interface.

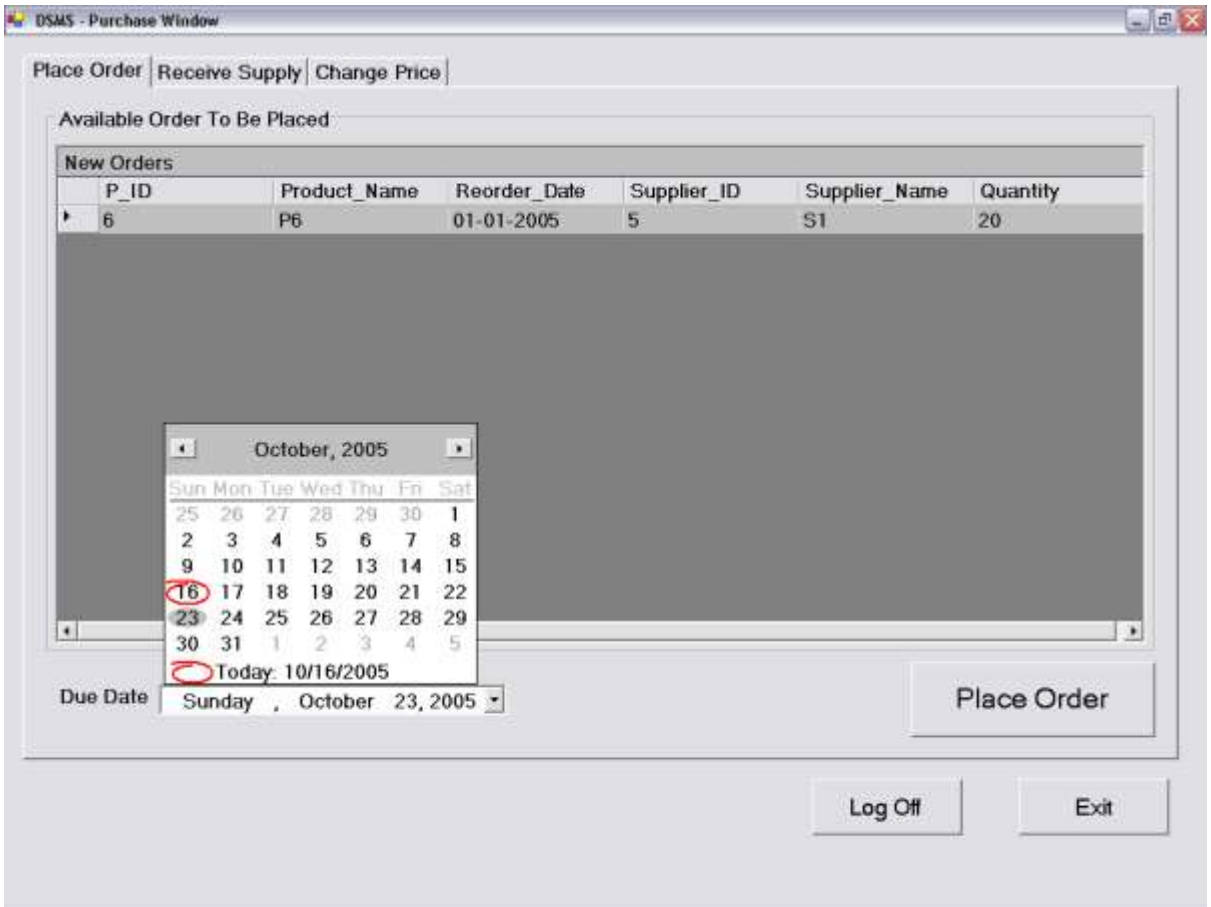

Figure #5: Departmental Store purchaser Interface

We summarize DSMS purchaser's functionality below:

Place Order:

Functionalities related to placing the order made by the admin are grouped under the '**Place Order**' tab. Here the purchaser can select the *available* orders from 'New Orders' table and select the due date.

Receive Supply:

Functionalities related to receiving supply from the supplier are grouped under the '**Receive Supply**' tab. Here the purchaser can select a previous order from 'Supply Orders' table and set the purchase price, selling price, batch code, mfg. date and expiration date of the current item of a product.

Change Price:

All the functionalities related to changing the selling price of different items of a product are grouped under the '**Change Price**' tab. Here the purchaser can select product id from product id dropdown list and then select an available item of that product and change its selling price to reflect the current market price of that product.

PART 3

Development Issues

Since this project was assigned as a term project for our Database Sessional Course (Course No CSE 304N), we had to do quite a bit of paper and pencil planning and design before we started coding. In the following sections we describe the tables, language and debugging tools we used for developing the software.

Tables:

We list the database tables and their attributes below:

Sl.	Table Name	Attributes
1.	BUSINESS_ENTITY	*BE_Id*, *BE_Name*, BE_Address
2.	PRODUCT_CATEGORY	*Category_Id*, *Parent_Category*, Sub_Category
3.	PRODUCT	*Product_Id*, *Product_Name*, *Manufacturer_Id*, *Category_Id*, *Unit_Desc*, *Reorder_threshold*
4.	ITEM	*Product_Id*, *Batch_No*, *Quantity*, *Current_Unit_Price*, Mfg_Date, Expr_Date
5.	INVOICE	*Invoice_No*, *Customer_Id*, *Sale_Date*
6.	INVOICE_ITEM	*Invoice_No*, *Product_Id*, *Batch_No*, *Sold_Unit_Price*, *Quantity*
7.	SUPPLIER	*Supplier_Id*, *Product_Id*, *Ratings*
8.	SUPPLY_ORDER	*Supply_Order_No*,*Supplier_Id*, *Product_Id*, Batch_No , *Quantity*, Purchased_Unit_Price, *Submission_Date*, *Due_Date*, Received_Date, *Status*
9.	EMPLOYEE	*Employee_Id*, *Father_Name*, *Mother_Name*, *Date_OF_Birth*, *Religion*, *Marital_Status*, *Parmanent_Address*, *Sex*, Photo, *Joining_Date*, Resigning_Date, *Salary*, *Job_Title*, *Status*, Experience, *Referred_By*
10.	SOLD_BY	*Invoice_No*, *Employee_Id*
11.	PURCHASED_BY	*Supply_Order_No*, *Employee_Id*
12.	USER_INFO	*User_Id*, *Employee_Id*, *Password*, *Privilege*

Sl.	Table Name	Attributes
13.	USER_LOG	*User_Id, Log_In_Time*, Log_Off_Time
14.	REORDER	*Product_Id*, *Reorder_Status*, *Reorder_Date*, Supplier_Id, *Quantity*
15.	EXPENSE	*Expense_Id*, *Expense_Amount*, *Employee_Id*, *Occurence*, *Description*
16.	DAMAGE	*Damage_Id*, Product_Id, Batch_No, *Damaged_On* , *Damage_Cause*, Damaged_By, *Quantity*, *Damage_Amount*, *Description*

N.B.:

- The attributes written in italic form are NOT NULL.
- The underlined attributes indicate the primary key of the table.
- The referenced attributes are understandable intuitively from naming.

Development Tools & Languages:

We primarily used two software tools for developing this software. These two tools can be categorized as follows:

- Front-end: Microsoft Visual Studio .NET 2003
- Backend: Oracle 10g

Since this software is application based and assumed to run on a windows platform, we decided to use C# (C-Sharp) as the implementaion language. Naturally, Microsoft Visual Studio .NET 2003 was the IDE to use for coding in C#. Graphical User Interface (GUI) was created the amazing gui developing tool built-into VS.net. We used .NET Framework 1.1. for developing this software which is the default version of .net framework installed with VS.net 2003. We used 'Oracle Client API' for connecting to the database and storing, retieving and updating the data in the database from the application.

We used Oracle 10g for the development process. We typed the code in DSMS.sql file (see appendix) in Oracle SQL*PLUS Worksheet and built the table schemas from there. Oracle managed the database and C# only interacted with it to store and retrieves the required data.

Debugging Tools:

Debugging is very crucial for any software development project. Our project was no exception. During developing the software, debugging was necessary because errors often occurred during executing commands to send and receive data. For debugging we used C# Debugger, Oracle SQL*PLUS Worksheet and Oracle Enterprise Manager Console.

A *good* debugger can reduce a programmer's hard work by half. ***MS Visual Studio.net*** has a very informative and friendly user interface for debugging the bugs in C# code. It has a separate DEBUG menu option in which it has many integrated options. One of which is very helpful is called 'Breakpoints' that enables us to execute a specific line or block of code for error. The line that is added to breakpoints is immediately enlightened. We have also used 'Step over' and 'Step in' options in conjunction with the breakpoints to check the errorneous code. The 'AUTOS' window enables us to see the current state of various objects and variables. This is extremely helpful for debugging because it enables us to see what went wrong and where it actually occurred. Moreover we can observe any specific variable or the state of the stack. All we need to do is to insert break points at positions where we think that an error or exceptions might occur.

The part of the code which was most difficult to debug was the construction of the queries at runtime by concatenation of strings and values of different variables. We used MessageBox to display the resultant query during the development process and corrected the errors.

The Oracle SQL*Plus **`show errors`** command displays all the errors associated with the most recently created procedural object. This command checks the `USER_ERRORS` data dictionary view for the errors associated with the most recent compilation attempt for that object. **`show errors`** displays the line and column number for each error, as well as the text of the error message. To view errors associated with previously created procedural objects, we queried USER_ERRORS directly. Two other data dictionary view levels – `ALL` and `DBA` – may also be used to retrieve information about errors involving procedural objects. In addition to the debugging information provided by the **`show errors`** command, we also used the DBMS_OUTPUT package. To use DBMS_OUTPUT, one must issue the **`set serveroutput on`** command before executing the procedural object one will be debugging.

Oracle also provides Enterprise Manager Console that enables us to compile queries, procedures, functions, triggers and it also offers us detailed information about the exceptions or compilation errors that might occur. In this way we can debug the exceptions and handle the errors that might occur.

PART 4

Last Words

Special Windows Forms Controls Used:

We used a special windows forms control called 'RegexTextbox' which is a derived class of Windows.Forms.TextBox class. It has the function of determining whether a given input matches the desired input format and rules. For this we have to give a regular expression in its property and enable its cause validation property. This property is built into ASP.net section of .NET Framework but is not available for application software development.

Future Plans:

Initially we had many features in our mind that we can add in this software but had to omit those later due to time limitation. We list some those features below:

- The supplies whose deliveries are delayed beyond an allowable time limit can be auto generated and be seen from Administrator's Supply interface.
- Searching and editing employee information by administrator. Improving the employee management capability of the administrator.
- Searching suppliers by administrator.
- Using crystal reports to print invoices and generate Daily, Monthly reports about profit, inventory, sales, purchases, purchase returns, damages etc.
- Currently payment can made only in cash .We can handle payment by Credit Cards. In order to do that, we must have interconnection with different banks. A server would receive information/conformation from the bank and sent it to the front-end. Request to the banks will be send from the front-end. We can also handle payment by money order.
- We can also add support for home delivery service.
- The system may have a website to allow customers to search the products they want with restricted information.

- We can add support for different branches of a departmental store and use networking for connecting all the branches.

- Building a complete and user-friendly help system for the users of this software.

APPENDIX

DSMS.SQL

The following text is the complete code listing we used in building the database for DSMS. If you are using *Oracle 10g* or later, then just type the following code in "SQL*PLUS Worksheet" and press F5. If you are using *Oracle 9i* or any previous version of Oracle, then omit the code block titled "**Purge Tables**" and type the rest of the following code in "SQL*PLUS Worksheet" and press F5. Your Database will be created!!!

```
--DSMS.SQL-----------------------------------------------------

--Purge Tables-------------------------------------------------
drop table DAMAGE purge;
drop table EXPENSE purge;
drop table REORDER purge;
drop table USER_LOG purge;
drop table USER_INFO purge;
drop table PURCHASED_BY purge;
drop table SOLD_BY purge;
drop table EMPLOYEE purge;
drop table SUPPLY_ORDER purge;
drop table SUPPLIER purge;
drop table INVOICE_ITEM purge;
drop table INVOICE purge;
drop table ITEM purge;
drop table PRODUCT purge;
drop table PRODUCT_CATEGORY purge;
drop table BUSINESS_ENTITY purge;

--BUSINESS_ENTITY Table-----------------------------------------
drop table BUSINESS_ENTITY;
create table BUSINESS_ENTITY(
BE_Id             NUMBER(10)        NOT NULL,
BE_Name           VARCHAR2(50)      NOT NULL,
BE_Address        VARCHAR2(100)     ,
constraint BUSINESS_ENTITY_PK primary key (BE_Id)
);

--PRODUCT_CATEGORY Table----------------------------------------
drop table PRODUCT_CATEGORY;
create table PRODUCT_CATEGORY(
Category_Id       NUMBER(10)        NOT NULL,
Parent_Category   VARCHAR2(30)      NOT NULL,
Sub_Category      VARCHAR2(30)      ,
constraint PRODUCT_CATEGORY_PK primary key (Category_Id)
);
```

--PRODUCT Table----------

```
drop table PRODUCT;
create table PRODUCT(
Product_Id        NUMBER(10)        NOT NULL,
Product_Name      VARCHAR2(30)      NOT NULL,
Manufacturer_Id   NUMBER(10)        NOT NULL,
Category_Id       NUMBER(10)        NOT NULL,
Unit_Desc         VARCHAR2(30)      NOT NULL,
Reorder_threshold NUMBER(10)        NOT NULL,
constraint PRODUCT_PK primary key (Product_Id),
constraint PRODUCT_FK1 foreign key (Manufacturer_Id) references
BUSINESS_ENTITY(BE_Id) on delete cascade,
constraint PRODUCT_FK2 foreign key (Category_Id) references
PRODUCT_CATEGORY(Category_Id) on delete cascade
);
```

--ITEM Table----------

```
drop table ITEM;
create table ITEM(
Product_Id        NUMBER(10)        NOT NULL,
Batch_No          NUMBER(10)        NOT NULL,
Quantity          NUMBER(10)        NOT NULL,
Current_Unit_Price NUMBER(10, 2)    NOT NULL,
Mfg_Date          DATE              ,
Expr_Date         DATE              ,
constraint ITEM_PK primary key (Product_Id, Batch_No),
constraint ITEM_FK foreign key (Product_Id) references PRODUCT(Product_Id)
on delete cascade
);
```

--INVOICE Table----------

```
drop table INVOICE;
create table INVOICE(
Invoice_No        NUMBER(10)        NOT NULL,
Customer_Id       NUMBER(10)        NOT NULL,
Sale_Date         DATE              NOT NULL,
constraint INVOICE_PK primary key (Invoice_No),
constraint INVOICE_FK foreign key (Customer_Id) references
BUSINESS_ENTITY(BE_Id) on delete cascade
);
```

--INVOICE_ITEM Table----------

```
drop table INVOICE_ITEM;
create table INVOICE_ITEM(
Invoice_No        NUMBER(10)        NOT NULL,
Product_Id        NUMBER(10)        NOT NULL,
Batch_No          NUMBER(10)        NOT NULL,
Sold_Unit_Price   NUMBER(10, 2)     NOT NULL,
Quantity          NUMBER(10)        NOT NULL,
constraint INVOICE_ITEM_PK primary key (Invoice_No, Product_Id, Batch_No),
constraint INVOICE_ITEM_FK1 foreign key (Invoice_No) references
INVOICE(Invoice_No) on delete cascade,
constraint INVOICE_ITEM_FK2 foreign key (Product_Id, Batch_No) references
ITEM(Product_Id, Batch_No) on delete cascade
);
```

--SUPPLIER Table----------

```
drop table SUPPLIER;
create table SUPPLIER(
```

```
Supplier_Id       NUMBER(10)        NOT NULL,
Product_Id        NUMBER(10)        NOT NULL,
Ratings           NUMBER(10)        NOT NULL check(Ratings <= 5 and Ratings >= 1),
constraint SUPPLIER_PK  primary key (Supplier_Id, Product_Id),
constraint SUPPLIER_FK1 foreign key (Supplier_Id) references
BUSINESS_ENTITY(BE_Id) on delete cascade,
constraint SUPPLIER_FK2 foreign key (Product_Id) references
Product(Product_Id) on delete cascade
);
```

--SUPPLY_ORDER Table----------

```
drop table SUPPLY_ORDER;
create table SUPPLY_ORDER(
Supply_Order_No   NUMBER(10)        NOT NULL,
Supplier_Id       NUMBER(10)        NOT NULL,
Product_Id        NUMBER(10)        NOT NULL,
Batch_No          NUMBER(10)        ,
Quantity          NUMBER(10)        NOT NULL,
Purchased_Unit_Price NUMBER(10, 2)  ,
Submission_Date   DATE              NOT NULL,
Due_Date          DATE              NOT NULL,
Received_Date     DATE              ,
Status            NUMBER(10)        NOT NULL check(Status <= 3 and Status >= 1),
constraint SUPPLY_ORDER_PK primary key (Supply_Order_No),
constraint SUPPLY_ORDER_FK1 foreign key (Supplier_Id) references
BUSINESS_ENTITY(BE_Id) on delete cascade,
constraint SUPPLY_ORDER_FK2 foreign key (Product_Id, Batch_No) references
ITEM(Product_Id, Batch_No) on delete cascade
);
```

--EMPLOYEE Table----------

```
drop table EMPLOYEE;
create table EMPLOYEE(
Employee_Id       NUMBER(10)        NOT NULL,
Father_Name       VARCHAR2(50)      NOT NULL,
Mother_Name       VARCHAR2(50)      NOT NULL,
Date_OF_Birth     DATE              NOT NULL,
Religion          VARCHAR2(30)      NOT NULL,
Marital_Status    VARCHAR2(30)      NOT NULL,
Parmanent_Address VARCHAR2(30)      NOT NULL,
Sex               VARCHAR2(1)       NOT NULL,
Photo             BFILE             ,
Joining_Date      DATE              NOT NULL,
Resigning_Date    DATE              ,
Salary            NUMBER(10)        NOT NULL,
Job_Title         VARCHAR2(30)      NOT NULL,
Status            VARCHAR2(30)      NOT NULL, -- WORKING | ON_LEAVE | LEFT
Experience        VARCHAR2(100)     ,
Referred_By       VARCHAR2(30)      NOT NULL,
constraint EMPLOYEE_PK primary key (Employee_Id),
constraint EMPLOYEE_FK2 foreign key (Employee_Id) references
BUSINESS_ENTITY(BE_Id) on delete cascade
);
```

--SOLD_BY Table----------

```
drop table SOLD_BY;
create table SOLD_BY(
Invoice_No        NUMBER(10)        NOT NULL,
Employee_Id       NUMBER(10)        NOT NULL,
```

```
constraint SOLD_BY_PK primary key (Invoice_No, Employee_Id),
constraint SOLD_BY_FK1 foreign key (Invoice_No) references
INVOICE(Invoice_No) on delete cascade,
constraint SOLD_BY_FK2 foreign key (Employee_Id) references
EMPLOYEE(Employee_Id) on delete cascade
);
```

--PURCHASED_BY Table----------

```
drop table PURCHASED_BY;
create table PURCHASED_BY(
Supply_Order_No   NUMBER(10)        NOT NULL,
Employee_Id       NUMBER(10)        NOT NULL,
constraint PURCHASED_BY_PK primary key (Supply_Order_No, Employee_Id),
constraint PURCHASED_BY_FK1 foreign key (Supply_Order_No) references
Supply_Order(Supply_Order_No) on delete cascade,
constraint PURCHASED_BY_FK2 foreign key (Employee_Id) references
EMPLOYEE(Employee_Id) on delete cascade
);
```

--USER_INFO Table----------

```
drop table USER_INFO;
create table USER_INFO(
User_Id           VARCHAR2(30)      NOT NULL,
Employee_Id       NUMBER(10)        NOT NULL,
Password          VARCHAR2(30)      NOT NULL,
Privilege         VARCHAR2(30)      NOT NULL, -- ADMIN | SELLER | PURCHASER
constraint USER_INFO_PK primary key (User_Id),
constraint USER_INFO_FK foreign key (Employee_Id) references
EMPLOYEE(Employee_Id) on delete cascade
);
```

--USER_LOG Table----------

```
drop table USER_LOG;
create table USER_LOG(
User_Id           VARCHAR2(30)      NOT NULL,
Log_In_Time       DATE              NOT NULL,
Log_Off_Time      DATE              ,
constraint USER_LOG_PK primary key (User_Id, Log_In_Time),
constraint USER_LOG_FK foreign key (User_Id) references USER_INFO(User_Id)
on delete cascade
);
```

--REORDER Table----------

```
drop table REORDER;
create table REORDER(
Product_Id        NUMBER(10)        NOT NULL,
Reorder_Status    NUMBER(1)         NOT NULL,
Reorder_Date      DATE              NOT NULL,
Supplier_Id       NUMBER(10)        ,
Quantity          NUMBER(10)        NOT NULL,
constraint REORDER_PK primary key (Product_Id),
constraint REORDER_FK1 foreign key (Product_Id) references
Product(Product_Id) on delete cascade,
constraint REORDER_FK2 foreign key (Supplier_Id,Product_Id) references
SUPPLIER(Supplier_Id,Product_Id) on delete cascade
);
```

--EXPENSE Table----------

```
drop table EXPENSE;
```

```
create table EXPENSE(
Expense_Id        VARCHAR2(30)      NOT NULL,
Expense_Amount    NUMBER(10, 2)     NOT NULL,
Employee_Id       NUMBER(10)        NOT NULL,
Occurence         DATE              NOT NULL,
Description       VARCHAR(100)      NOT NULL,
constraint EXPENSE_PK primary key (Expense_Id),
constraint EXPENSE_FK1 foreign key (Employee_Id) references
EMPLOYEE(Employee_Id) on delete cascade
);
```

--DAMAGE Table----------

```
drop table DAMAGE;
create table DAMAGE(
Damage_Id         VARCHAR2(30)      NOT NULL,
Product_Id        NUMBER(10)        ,
Batch_No          NUMBER(10)        ,
Damaged_On        DATE              NOT NULL,
Damage_Cause      VARCHAR2(100)     NOT NULL,
Damaged_By        NUMBER(10)        ,
Quantity          NUMBER(10)        NOT NULL,
Damage_Amount     NUMBER(10, 2)     NOT NULL,
Description       VARCHAR(100)      NOT NULL,
constraint DAMAGE_PK primary key (Damage_Id),
constraint DAMAGE_FK1 foreign key (Product_Id, Batch_No) references
ITEM(Product_Id, Batch_No) on delete cascade,
constraint DAMAGE_FK2 foreign key (Damaged_By) references
EMPLOYEE(Employee_Id) on delete cascade
);
```

--INITIAL DATA----------

```
insert into BUSINESS_ENTITY values(0,'CASH', NULL);
insert into BUSINESS_ENTITY values(1,'NA', 'NA');
insert into BUSINESS_ENTITY values(2,'Admin', NULL);

insert into EMPLOYEE values(2, 'father', 'mother', to_date('1/1/1900',
'dd/mm/yyyy'), 'Islam', 'single', 'mars', 'm', NULL, SYS_DATE, NULL, 10000,
'M.D.', 'working', NULL, 'self');

insert into USER_INFO values('q',2,'q','admin');
insert into USER_INFO values('a',2,'a','seller');
insert into USER_INFO values('z',2,'z','purchaser');
```
