### CHAPTER 1

**INTRODUCTION**

* 1. **Introduction to DBMS with architecture diagram**

A **database** is an organized collection of data. A relational database, more restrictively, is a collection of schemas, tables, queries, reports, views, and other elements.

A database-management system (DBMS) is a computer-software application that interacts with end-users, other applications, and the database itself to capture and analyze data. A general-purpose DBMS allows the definition, creation, querying, update, and administration of databases. Well-known DBMSs include MySQL, PostgreSQL, EnterpriseDB, MongoDB, MariaDB, Microsoft SQL Server, Oracle, Sybase, SAP HANA, MemSQL, SQLite and IBM DB2.

Defining a database involves specifying the data types, structures and constraints of the data to be stored in the database. The database definition or descriptive information is also stores in the database in the form of a database catalog or dictionary; it is called meta-data. Constructing the database is the process of storing the data on some storage medium that is controlled by the DBMS. Manipulating a database includes functions such as querying the database to retrieve specific data, updating the database to reflect changes in the miniworld, and generating reports from the data. Sharing a database allows multiple users and programs to access the database simultaneously.

An application program accesses the database by sending queries or requests for data to the DBMS. A query typically causes some data to be retrieved; a transaction may cause some data to be read and some data to be written into the database.

Other important functions provided by the DBMS include protecting the database and maintaining it over a long period of time. Protection includes system protection against hardware or software malfunction and security protection against unauthorized or malicious access. A typical large database may have a life cycle of many years, so the DBMS must be able to maintainthe database system by allowing the system to evolve as requirements change over time.

The design of a DBMS depends on its architecture. It can be centralized or decentralized or hierarchical. The architecture of a DBMS can be seen as either single tier or multi-tier. An n-tier architecture divides the whole system into related but independent **n** modules, which can be independently modified, altered, changed, or replaced.

In 1-tier architecture, the DBMS is the only entity where the user directly sits on the DBMS and uses it. Any changes done here will directly be done on the DBMS itself. It does not provide handy tools for end-users. Database designers and programmers normally prefer to use single-tier architecture.

If the architecture of DBMS is 2-tier, then it must have an application through which the DBMS can be accessed. Programmers use 2-tier architecture where they access the DBMS by means of an application. Here the application tier is entirely independent of the database in terms of operation, design, and programming.

**3-tier Architecture**

A 3-tier architecture separates its tiers from each other based on the complexity of the users and how they use the data present in the database. It is the most widely used architecture to design a DBMS.



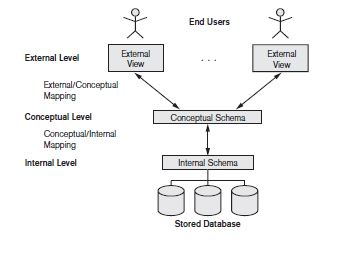


Fig 1.0. The database 3 tier architecture

* **Database (Data) Tier** − the database resides along with its query processing languages. We also have the relations that define the data and their constraints at this level.
* **Application (Middle) Tier** − the application server and the programs that access the database. For a user, this application tier presents an abstracted view of the database. End-users are unaware of any existence of the database beyond the application. At the other end, the database tier is not aware of any other user beyond the application tier. Hence, the application layer sits in the middle and acts as a mediator between the end-user and the database.
* **User (Presentation) Tier** − End-users operate on this tier and they know nothing about any existence of the database beyond this layer. At this layer, multiple views of the database can be provided by the application. All views are generated by applications that reside in the application tier.

Multiple-tier database architecture is highly modifiable, as almost all its components are independent and can be changed independently.

Entity-Relationship (ER) Model is based on the notion of real-world entities and relationships among them. While formulating real-world scenario into the database model, the ER Model creates entity set, relationship set, general attributes and constraints.

ER Model is best used for the conceptual design of a database.

ER Model is based on −

* **Entities** and their *attributes.*
* **Relationships** among entities.

These concepts are explained below.

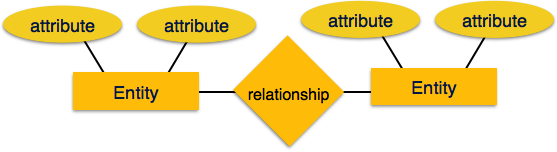


Fig. 1.1 Structure of an entity relationship model

* **Entity** – An entity in an ER Model is a real-world entity having properties called **attributes**. Every **attribute** is defined by its set of values called **domain**. For example, in a school database, a student is considered as an entity. Student has various attributes like name, age, class, etc.
* **Relationship** − the logical association among entities is called ***relationship***. Relationships are mapped with entities in various ways. Mapping cardinalities define the number of association between two entities.

Mapping cardinalities −

* + one to one
  + one to many
  + many to one
  + many to many

**Relational Model**

The most popular data model in DBMS is the Relational Model. It is more scientific a model than others. This model is based on first-order predicate logic and defines a table as an **n-ary relation**.

**DBMS Component Modules**

The database and the DBMS catalog are usually stored on disk. Access to the disk is controlled primarily by the operating system (OS), which schedules disk

Read/write. Many DBMSs have their own buffer management module to schedule disk read/write, because this has a considerable effect on performance. Reducing disk read/write improves performance considerably. A higher-level stored data manager module of the DBMS controls access to DBMS information that is stored on disk, whether it is part of the database or the catalog.

The DDL compiler processes schema definitions, specified in the DDL, and stores descriptions of the schemas (meta-data) in the DBMS catalog. The catalog includes information such as the names and sizes of files, names and data types of data items, storage details of each file, mapping information among schemas, and constraints.

In addition, the catalog stores many other types of information that are needed by the DBMS modules, which can then look up the catalog information as needed.

Application programmers write programs in host languages such as Java, C, or C++ that are submitted to a precompiler. The **precompiler** extracts DML commands from an application program written in a host programming language. These commands are sent to the DML compiler for compilation into object code for database access. The rest of the program is sent to the host language compiler. The object codes for the DML commands and the rest of the program are linked, forming a canned transaction whose executable code includes calls to the runtime database processor. Canned transactions are executed repeatedly by parametric users, who simply supply the parameters to the transactions. Each execution is considered to be a separate transaction.

It is now common to have the **client program** that accesses the DBMS running on a separate computer from the computer on which the database resides. The former is called the **client computer** running a DBMS client software and the latter is called the **database server**. In some cases, the client accesses a middle computer, called the **application server**, which in turn accesses the database server.

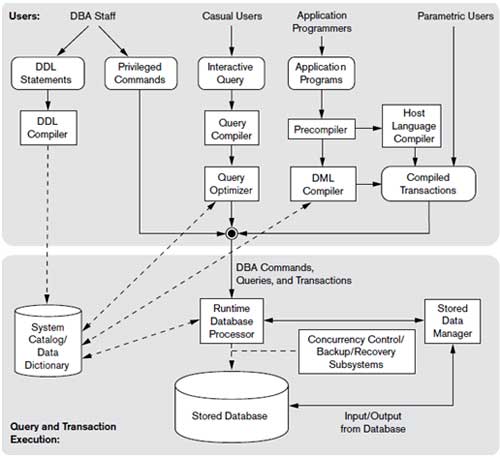


Fig. 1.3 Component modules of a DBMS and their interactions

**1.2 Overview of the Project**

The Pharmacy Management System, in all its uniqueness, is a comprehensive solution for efficient, quick and elegant management of a medical enterprise which includes a wide array of functional activities ranging from medicine and store management to supply and transaction processing.

* 1. **Problem Statement**

To build a Management System that can efficiently handle all the functional activities of a pharmaceutical enterprise- Store Management, drug management, transactions, distribution and supply.

**1.4 Objectives of the project**

1. To handle the data of all stores affiliated to a particular enterprise and means to setup records of an allocated store.
2. To handle the distribution of medicines to each store whereby a virtual representation of a dealer/distributor/supplier is considered.
3. To manage and monitor all the transactions associated with every store, and to be able to retrieve the details of the desired transactions.
4. To keep a constant track on the medicines available in each store, their expiry status, and the quantity. Necessary measures are to be taken as and when the above said status hits an undesirable mark.
5. To obtain useful statistical information about the entire system, (e.g.: The list of medicines that are the best sellers), which may provide a great impetus for the organizations to expand and improve their business, ultimately benefitting the society.
6. To provide security in terms of authentication so that only the authorized personnel is allowed to carry out his activities in his sphere of control.
7. To ensure interoperability between the various functional units which may be accomplished by coordinated messages and requests, while at the same time maintaining the integrity of each unit.

**CHAPTER 2**

**SYSTEM DESIGN AND METHODOLOGY**

**2.1 System Architecture**

**Softwares used:**

|  |  |
| --- | --- |
| SoftwareUsed | Version |
| Oracle | 10g Express Edition (XE) |
| JAVA | jdk1.8.0\_121 |
| NetBeans | 8.2 |

The Pharmacy Management System can be represented graphically as show in the block diagram.

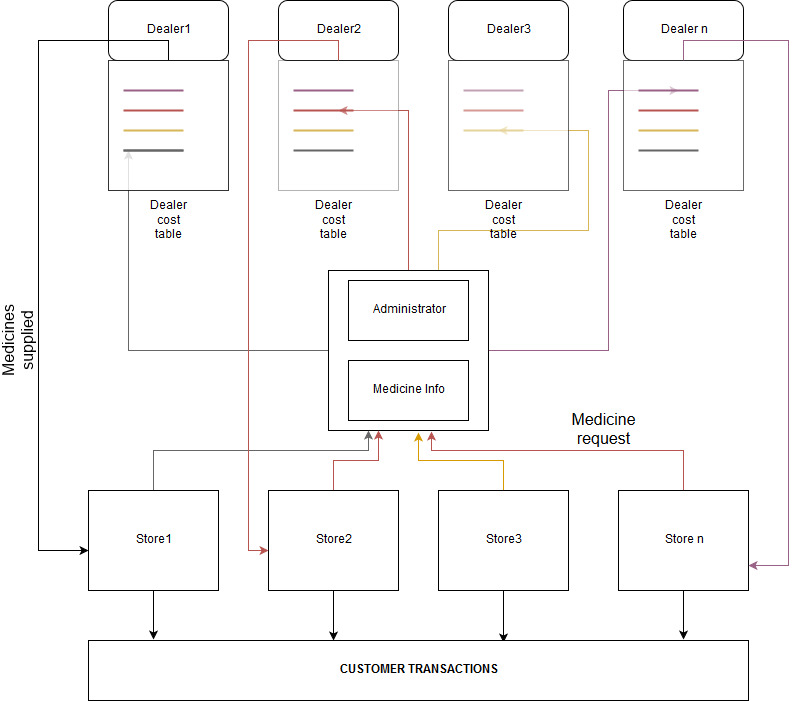


Fig.1.4 The architecture of the Pharmacy Management System

The Pharmacy Management System comprises of three main units:

1. Administrator
2. Stores
3. Dealer

1. The Administrator is the owner of the pharmaceutical enterprise and is solely responsible for the management of the system. The functions of the administrator are to:
2. Set up new records for a new store establishment, or for the removal a current establishment, and convey the authentication information to the respective store manager (as shown in Fig.1.4).
3. Announce the arrival of new medicines to all the stores under the enterprise
4. Update the information of medicines regularly
5. Set up records for a new dealer which the enterprise ascribes to, and convey the authentication information to the respective dealer
6. Update the dealer information regularly
7. Monitor the activities of stores, and keep a track on the transactions associated with every store that forms a part of the organization
8. To assign the requests of medicines from the stores to the dealers based on his discretion (usually referring to the cost table hosted by each dealer).
9. To analyze the statistical information gathered from all stores. Provisions are provided to view the details of a particular store or a dealer. For example, the administrator has to type in the dealer\_id, and the stores that he supplies to along with the quantity are retrieved.
10. Stores are independent establishments or openings, from where medicines are sold to the customers. The administrator can create any number of stores and assign each of these stores with login credentials. Each store is managed by a store manager, who keeps track of the medicines, the rack number, and the quantity of each medicine and the expiry status of the medicines. If falling short of a medicine (or if more stock is desired), the store has to initiate a request to the administrator.

The request comprises the medicine and the quantity necessary.

The administrator looks up all the dealers’ cost table (which shows the required medicine and the cost hosted by each dealer) and chose the dealer which he feels is viable to deal with. (The colored lines in Fig.1.4 represent distinct requests from each store).

After receiving the stock, the stores initiate transactions with the customer, and the details of the medicine bought, the quantity and the total amount is recorded and stored in the database. Appropriate triggers are used to alter the quantity of medicine in a store after a transaction between the store and a customer takes place or after a dealer supplies to a store.

1. The dealer represents the virtual link between the enterprise and the actual dealer. The dealer has two primary important functions. Intuitively,
2. The dealer is responsible for supplying the stores with the desired medicine and quantity. (The request for supply is directed to a dealer after the administrator chooses him for a particular medicine)
3. The dealer hosts his cost table which contains information about the medicines and the cost that he intends to charge (usually below the MRP of the medicine).

**2.2 ER Diagram:**

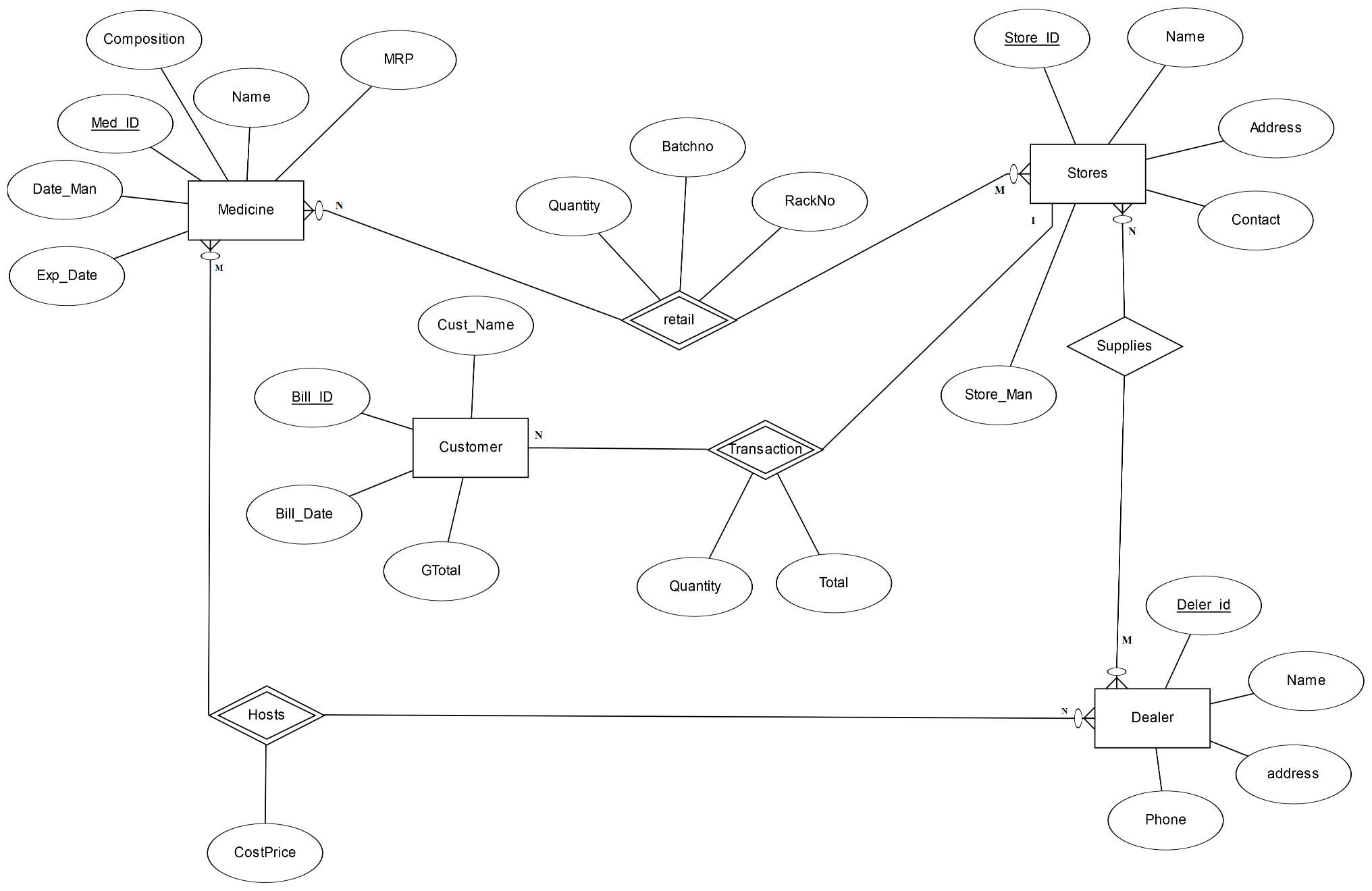


Fig.1.5 ER diagram of the Pharmacy Management System

**2.3 SCHEMA DIAGRAM**

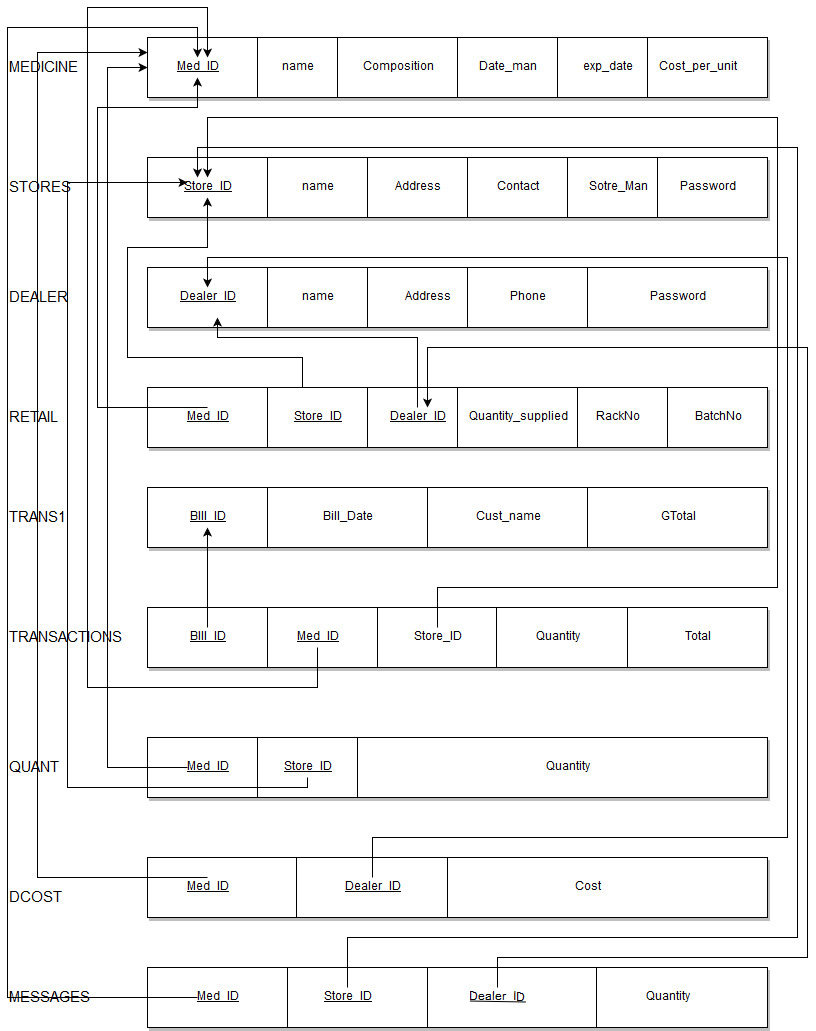


Fig. 1.6 the Schema

**CHAPTER 3**

**SYSTEM IMPLEMENTATION**

**3.1 MODULE DESCRIPTION**

**1. ADMINISTRATOR LOGIN**

**Objective of the module**

This enables the administrator to login into the system using his predefined password.

**Input**

* Password of the Administrator

**Output**

If the password entered is right, the appropriate message indicates that the login is successful, otherwise, login fails and a message is displayed

**Description**

The authentication is performed by comparing the password entered with the actual password. This password is fixed and is known only to the Administrator.

**2. RETAILER LOGIN**

**Objective of the module**

To enable the retailer to login with his credentials- store\_id and password

**Input**

* **Store\_id**
* **Password**

**Output**

If the password entered corresponds to the actual password for the store\_id, then the retailer login is successful. Appropriate messages are displayed depending on the correctness of the password.

**Description**

The entered password is compared with the password assigned to the store. This password is assigned to the store by the administrator at the time of store creation, and is stored in the database. The store manager may later change the password.

**3. DEALER LOGIN**

**Objective of the module**

To enable the retailer to login with his credentials- dealer\_id and password

**Input**

* Dealer\_id
* Password

**Output**

If the password entered corresponds to the actual password for the dealer\_id, then the retailer login is successful. Appropriate messages are displayed depending on the correctness of the password.

**Description**

The entered password is compared with the password assigned to the Dealer. This password is assigned to the dealer by the administrator at the time of dealer association, and is stored in the database. The dealer may later change the password.

**4. ADD NEW MEDICINE**

**Objective of the module**

To add a new medicine and the associated details to the database

**Input**

* Medicine\_id (med\_id)
* Name
* Composition
* Date of manufacture (Date\_man)
* Expiry Date (Exp\_Date)
* Cost\_Per\_Unit (MRP of the medicine)

**Output**

The above details entered by the user will be stored in the database, and an adjoining table on the screen displays the new record along with the existing records.

**Description**

This function is carried out only by the administrator. Semantically, it conveys the arrival of a new medicine into the market.

**5. UPDATE MEDICINE**

**Objective of the module**

Updating of information associated with a particular medicine

**Input**

* Name
* Composition
* Date of manufacture (Date\_man)
* Expiry Date (Exp\_Date)
* Cost\_Per\_Unit (MRP of the medicine)

**Output**

The details entered by the user are recorded and are updated to the specific tuple that had to be updated. The user is required to click on the record (displayed in a table) that he wants to update. The details are transferred to text boxes where the user can update the information. Once the process is complete, the update button is pressed, and the updated medicine is displayed on an adjoining table.

**5. DELETE MEDICINE**

**Objective of the module**

Deleting all records associated with a specified medicine uniquely identified by its med\_ID

**Input**

* The user is required to mouse-click on the medicine (displayed on a table) that he wants to remove.
* The selected row is removed by clicking on the ‘remove medicine’ button

**Output**

All records and information associated with that medicine identified by its med\_ID are permanently deleted from the database, which is also reflected in the table displayed to the user.

**6. ADD NEW STORE**

**Objective of the module**

To set up a new store and establish the details of the newly set up store

**Input**

* Name
* Address
* Contact
* Store Manager
* Password

**Output**

The above details entered by the user will be stored in the database, with the value of store\_id being automatically generated and an adjoining table on the screen displays the new record along with the existing records.

**Description**

This function is carried out only by the administrator. The password assigned to the store is conveyed to the store manager- who logs into his store initially with this password and can change it later on.

**6. UPDATE STORE**

**Objective of the module**

Updating of information associated with a particular store

**Input**

* Name
* Address
* Contact
* Store manager

**Output**

The details entered by the user are recorded and are updated to the specific tuple that had to be updated. The user is required to click on the record (displayed in a table) that he wants to update. The details are transferred to text boxes where the user can update the information. Once the process is complete, the update button is pressed, and the updated store is displayed on an adjoining table.

**7. DELETE STORE**

**Objective of the module**

Deleting all records associated with a specified store uniquely identified by its store\_ID

**Input**

* The user is required to mouse-click on the store (displayed on a table) that he wants to remove.
* The selected row is removed by clicking on the ‘delete’ button

**Output**

All records and information associated with the store identified by its store\_ID are permanently deleted from the database, which is also reflected in the table displayed to the user.

**8. ADD A NEW DEALER**

**Objective of the module**

To set up a contract with a new dealer and establish the details of the newly set up dealer

**Input**

* Dealer\_id
* Name
* Address
* Contact
* Password

**Output**

The above details entered by the user will be stored in the database, and an adjoining table on the screen displays the new record along with the existing records.

**Description**

This function is carried out only by the administrator. The password assigned to the dealer is conveyed to the dealer- who logs in store initially with this password and can change it later on.

**9. UPDATE DEALER**

**Objective of the module**

Updating of information associated with a particular dealer

**Input**

* Name
* Address
* Contact
* password

**Output**

The details entered by the user are recorded and are updated to the specific tuple that had to be updated. The user is required to click on the record (displayed in a table) that he wants to update. The details are transferred to text boxes where the user can update the information. Once the process is complete, the update button is pressed, and the updated details of the dealer are displayed on an adjoining table.

**10. DELETE DEALER**

**Objective of the module**

Deleting all records associated with a specified dealer uniquely identified by his dealer\_ID

**Input**

* The user is required to mouse-click on the dealer (displayed on a table) that he wants to remove.
* The selected row is removed by clicking on the ‘delete’ button

**Output**

All records and information associated with the dealer identified by his dealer\_ID are permanently deleted from the database, which is also reflected in the table displayed to the user.

**11. VIEW TRANSACTIONS AS AN ADMINISTRATOR**

**Objective of the module**

Enabling the administrator to view transactions performed by every store that is affiliated to his organization

**Input**

* The user is required to mouse-click on the desired record displayed on the table that displays all transactions from all stores
* The bill\_id is retrieved and the submit button is to be pressed
* The bill\_id may also be manually typed

**Output**

All the details associated with that transaction- customer name, bill\_id, bill\_date, store\_id, name, the medicines bought along with their names and quantities, and the grand total is displayed on the screen. In other words a bill is displayed on the screen

**Description**

The administrator is authorized to view any transactions made by any store affiliated to his organization

**12. REQUEST FOR SUPPLY**

**Objective of the module**

To place a request to the administrator enlisting the medicine and the quantity required by a particular store which is currently logged in

**Input**

* Medicine id
* Quantity

**Output**

The verification message that the request has been placed is displayed and the request is stored in the database

**Description**

The store\_id is recorded as soon as the store keeper/manager logs in. This information is vital to determine for which store the medicines ought to be supplied to. The med\_id is obtained by just clicking on the item displayed on a table or can be entered manually. Additional features like revert are added for convenient placement of requests. This request will be available in the inbox section of the administrator. A request is automatically placed if the quantity goes below a particular value

**13. INITIATE TRANSACTIONS**

**Objective of the module**

To begin a transaction with a customer, and to produce a bill that contains all the details of that transaction

**Input**

* Medicine ID
* Quantity required
* Customer name

**Output**

The contents are updated in the database and a verification message that a bill has been uploaded successfully to the database, is displayed.

**Description**

The store\_id is recorded as soon as the store keeper/manager logs in. This information is vital to determine for which store the medicines are transacted. The med\_id is obtained by just clicking on the item displayed on the table or can be entered manually. The quantity required by the customer should be entered manually. The bill\_date is retrieved from the system’s today’s date. Additional features like revert are added for convenient transactions.

**14. VIEW TRANSACTIONS AS A STORE-KEEPER**

**Objective of the module**

To allow a particular store-keeper, identified by his store’s store\_id, to view transactions that have been initiated by that store

**Input**

* Bill\_id

**Output**

All the details associated with that transaction- customer name, bill\_id, bill\_date, store\_id, name, the medicines bought along with their names and quantities, and the grand total is displayed on the screen. In other words a bill is displayed on the screen

**Description**

Note that this transaction view is different from that of an administrator. The administrator gets access to all transactions from all stores. The store keeper gets access to transactions of only his store. When the store keeper logs in, his credentials are recorded. The store\_id is a vital information for handling all store operations.

**15. CHANGE THE STORE PASSWORD**

**Objective of the module**

To enable the store-keeper/manager to change the password for his store’s store\_id, so that no other unauthorized personnel can login with malicious intent.

**Input**

* The new password that the store-keeper assigns

**Output**

The password is changed and updated in the database and an appropriate message is displayed

**Description**

When the store-keeper/manager logs in for the next time, he has to enter the new password. This feature is also available to the dealer

**16. DEALER COST TABLE**

**Input**

* The medicine id
* The cost that the dealer wants to assign for that medicine

**Output**

The medicine\_id and their corresponding cost hosted by the dealer identified by the dealer\_id is displayed on an adjoining table.

Description

When the dealer logs in with his dealer\_id the dealer\_id is recorded. This information is vital to all operations related to the dealer. The resultant cost table is referred to by the administrator who decides if this dealer’s offer has to be considered.

**17. SUPPLY OF MEDICINES TO THE STORES**

**Objective of the module**

To supply the required quantity of medicine to the destined store as per the request sent in by the administrator

**Input**

* Medicine ID
* Store ID
* Quantity required

**Output**

A message conforming that the supply has been arranged is displayed on the screen. The stores are updated with the required quantity of the desired medicines.

**Description**

The medicine ID, store ID and the quantity required are obtained by directly clicking on the requests displayed on the table. After the supply is arranged for a particular store, the change in quantity of that medicine can be observed when the store that is supplied to is logged into.

**18. DEALER ASSIGNMENT FOR REQUESTS**

**Objectives of the module**

To select a dealer based on the administrator’s discretion referring to the cost table, and assign the request to that dealer identified by the dealer\_id

**Input**

* Medicine ID
* Store ID
* Quantity
* Dealer\_ID

**Output**

The request enlisting the medicines with the quantity, from a store identified by store\_id is sent to the dealer identified by the dealer\_id.

**Description**

The administrator performs this operation as it is based on his discretion that a dealer hosting a medicine is selected to process a request, out of other dealers hosting the same medicine.

**19. DEALERS SUPPLYING TO A STORE**

Objective

To view the dealers, and the medicines supplied to a particular store

**Input**

* Store\_ID

**Output**

The list of dealers supplying to that store are displayed on a table. Selecting a particular dealer by a mouse-click displays the medicines and the quantity supplied in the adjoining table

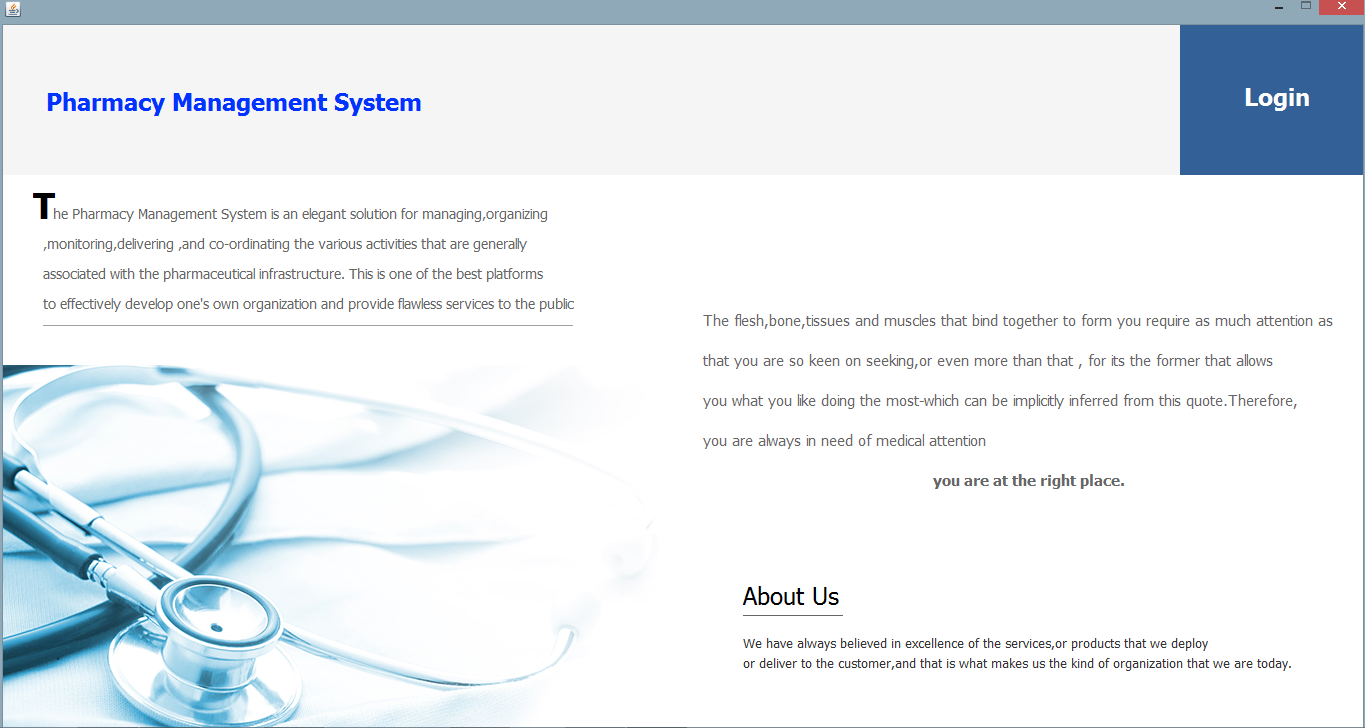
**Description**

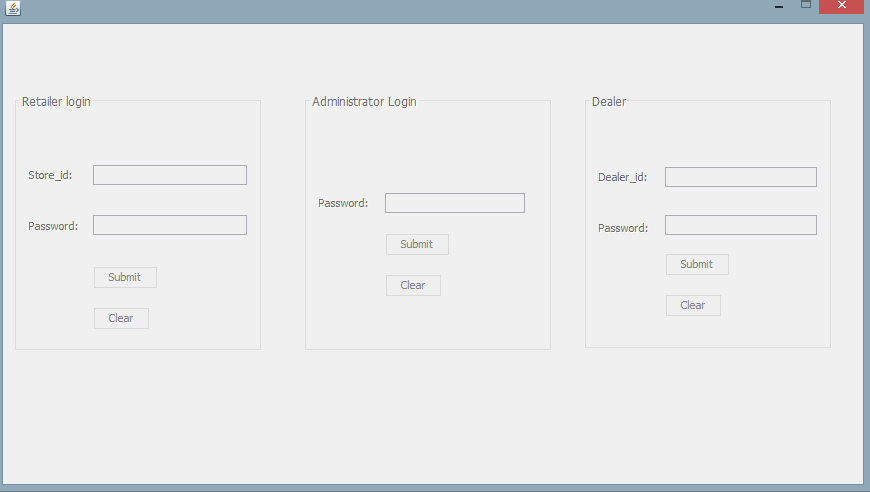
This power is vested only in the administrator. Similar statistical information is obtained by specifying store-id, dealer-id, med-id to view medicines in a store, stores supplied to by a dealer, stores that contain the medicine respectively.

**CHAPTER 4**

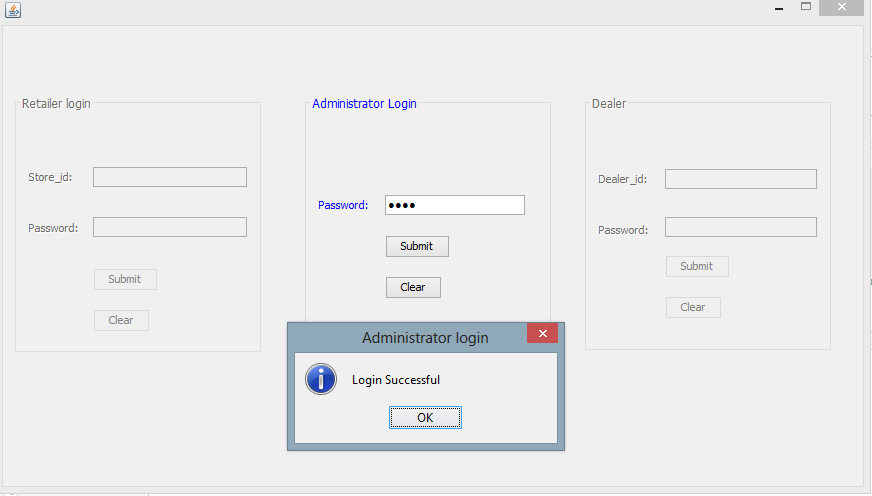
**RESULTS AND SCREENSHOTS**

**4.1 The Start Page**

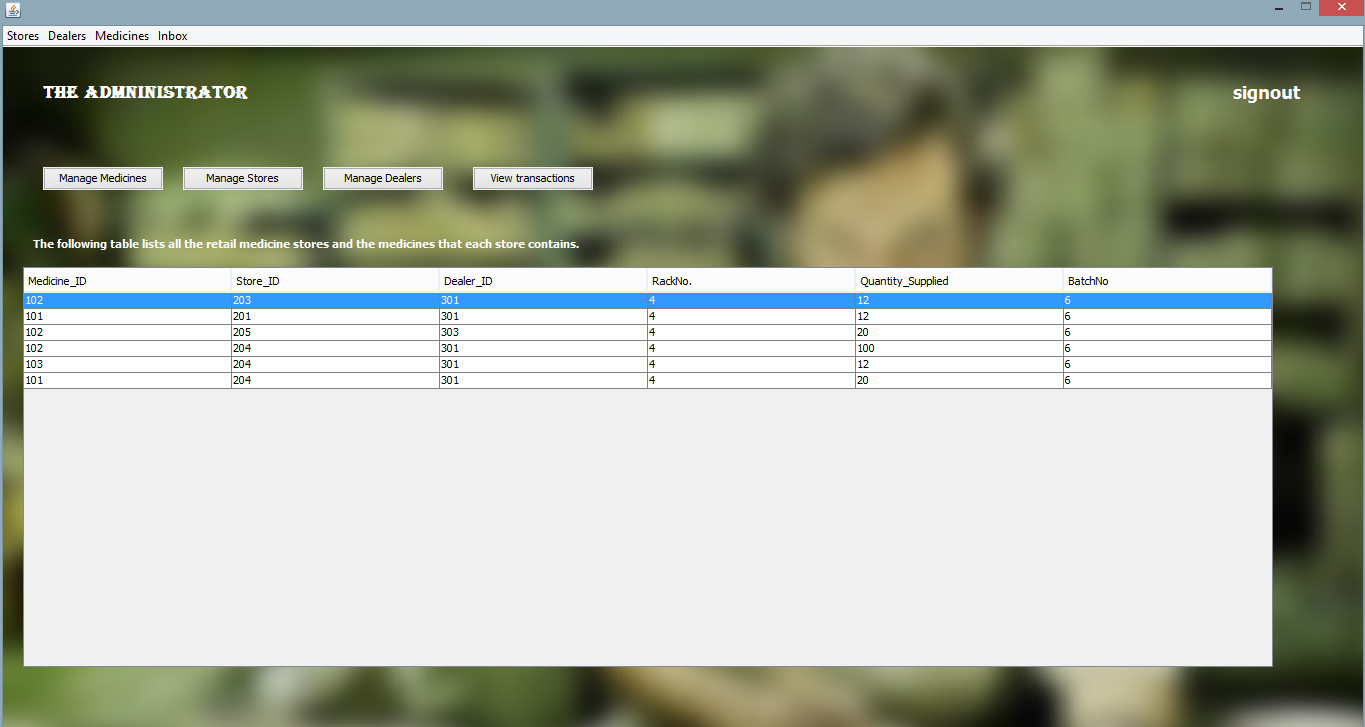


**4.2 Login Page**

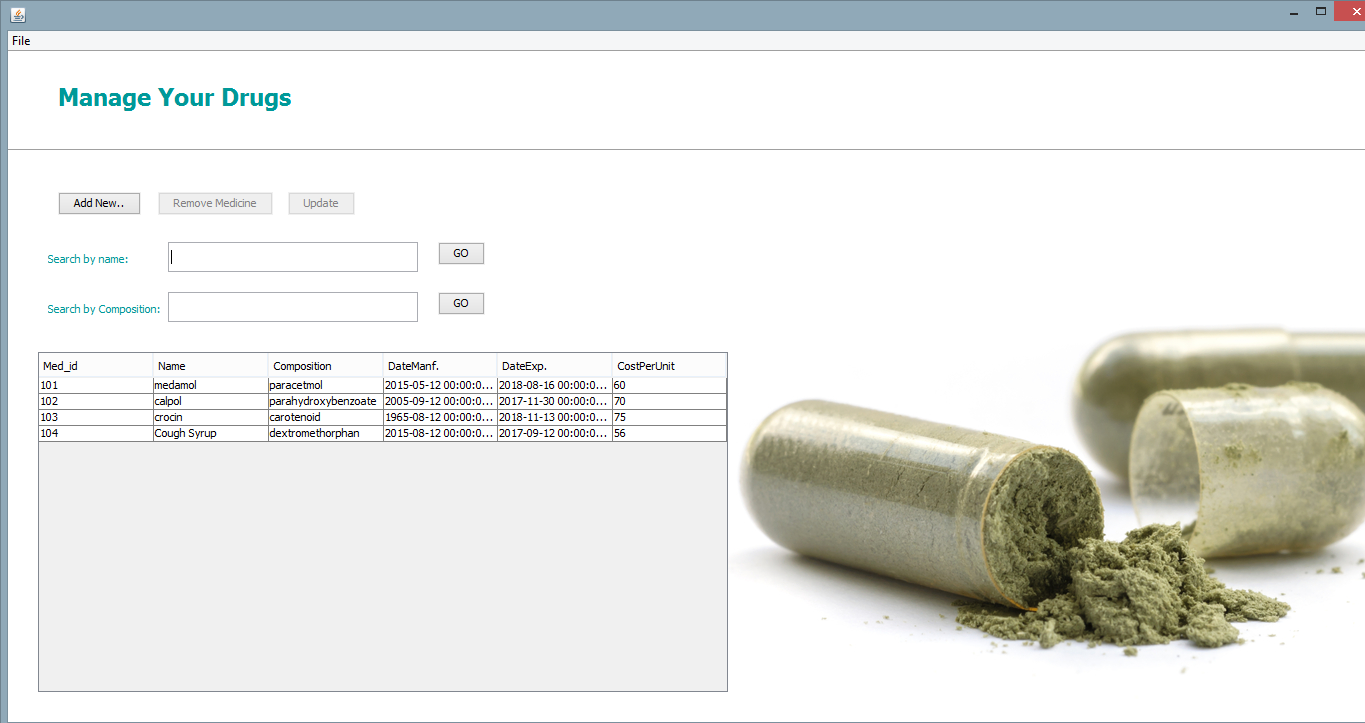
**4.3 Administrator login**



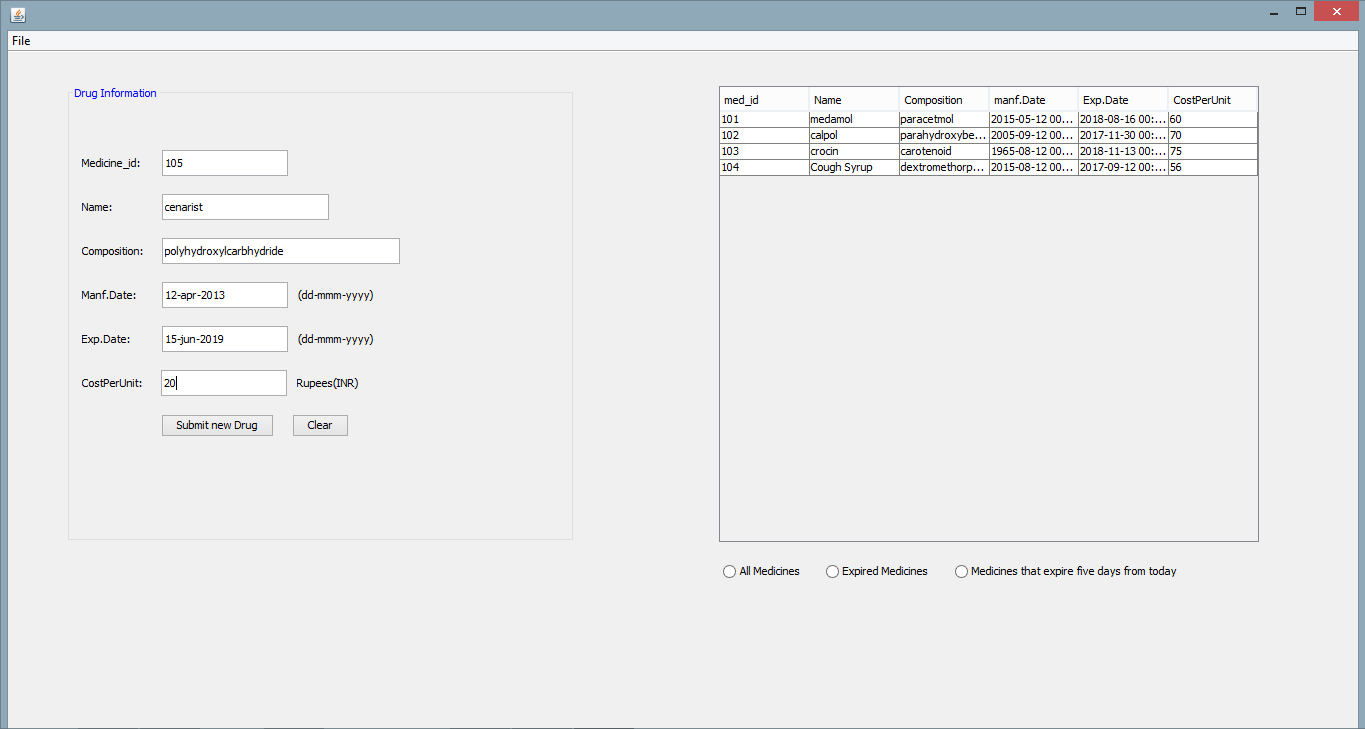
**4.4 The Administrator**



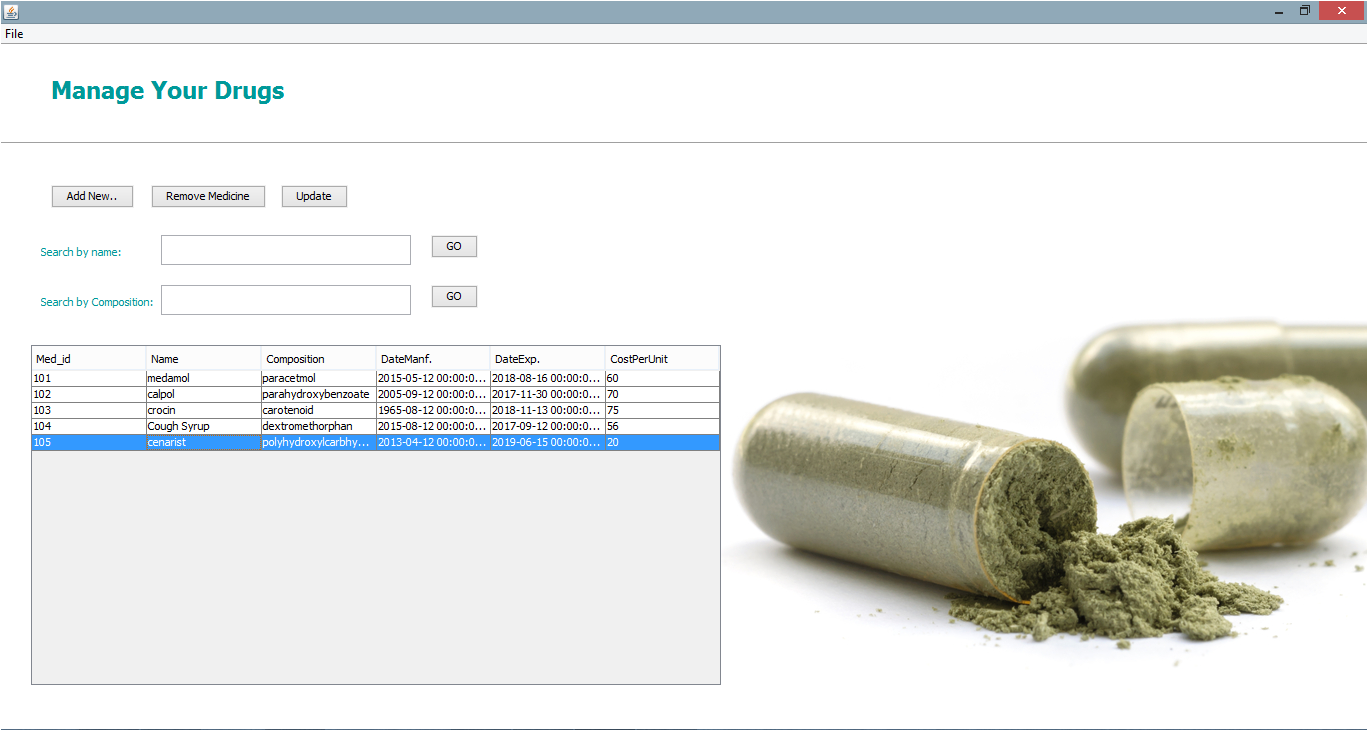
**4.5 Medicine Management**

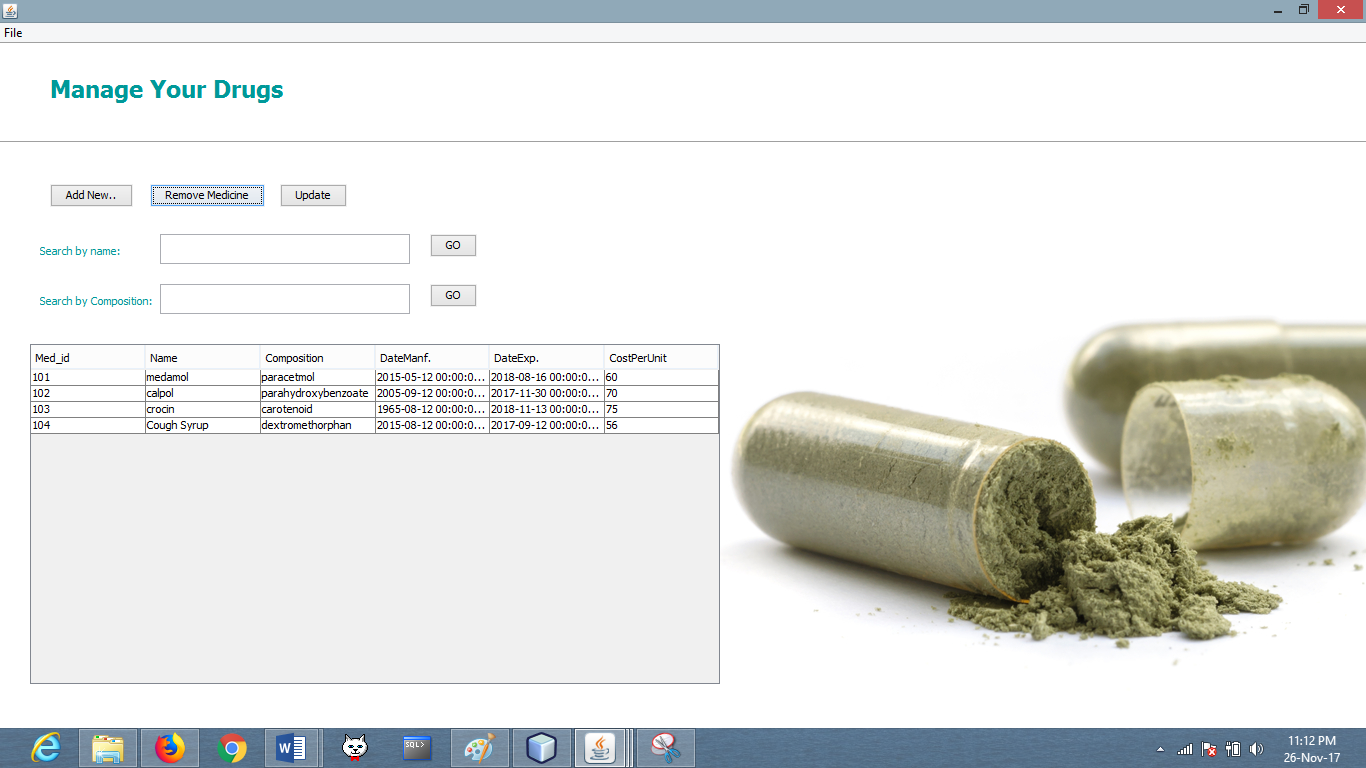


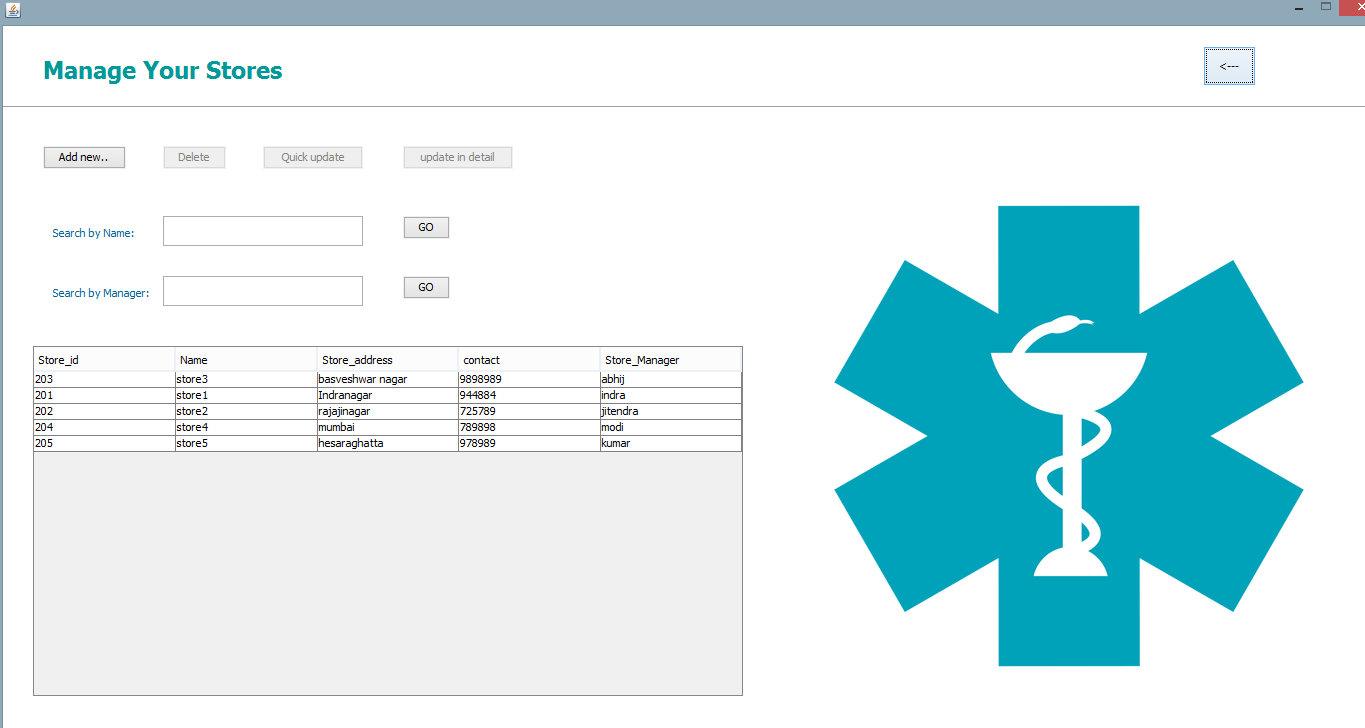
**4.6 Adding a new medicine**



**4.7 Deleting a medicine**

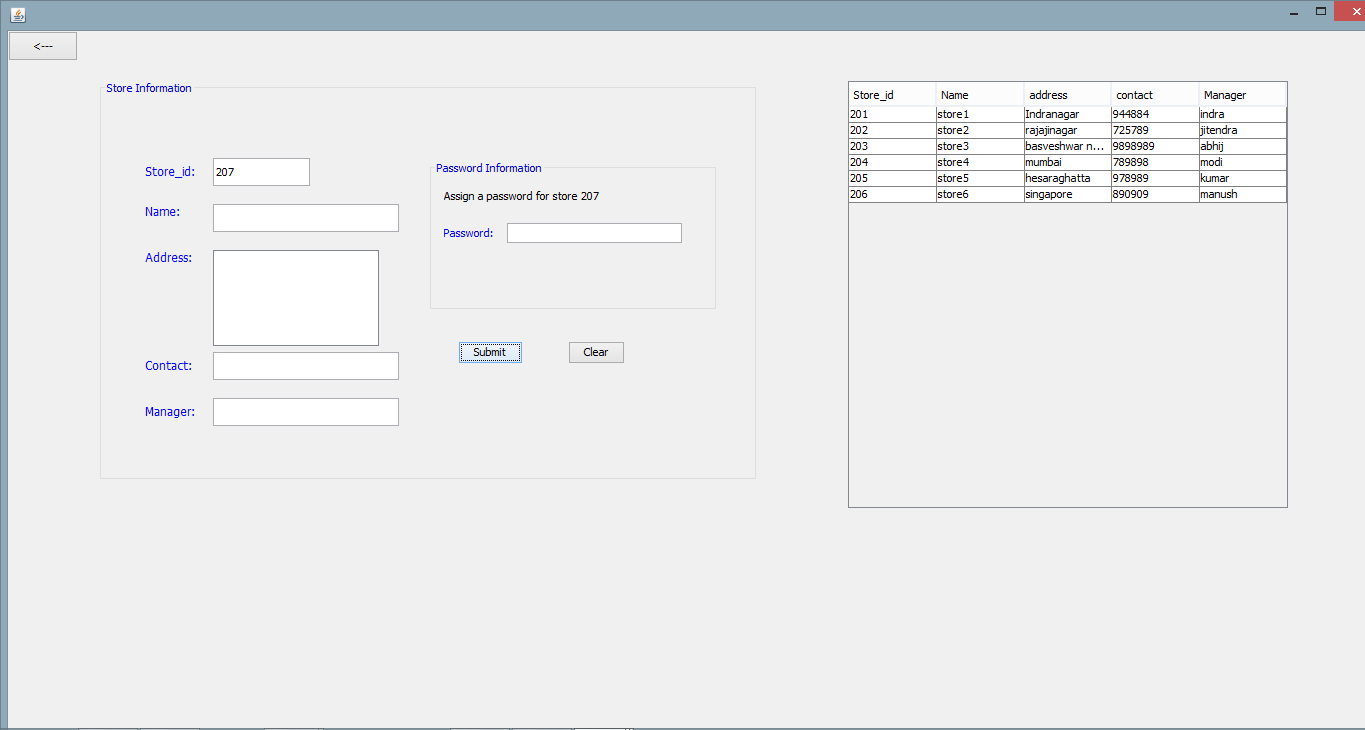


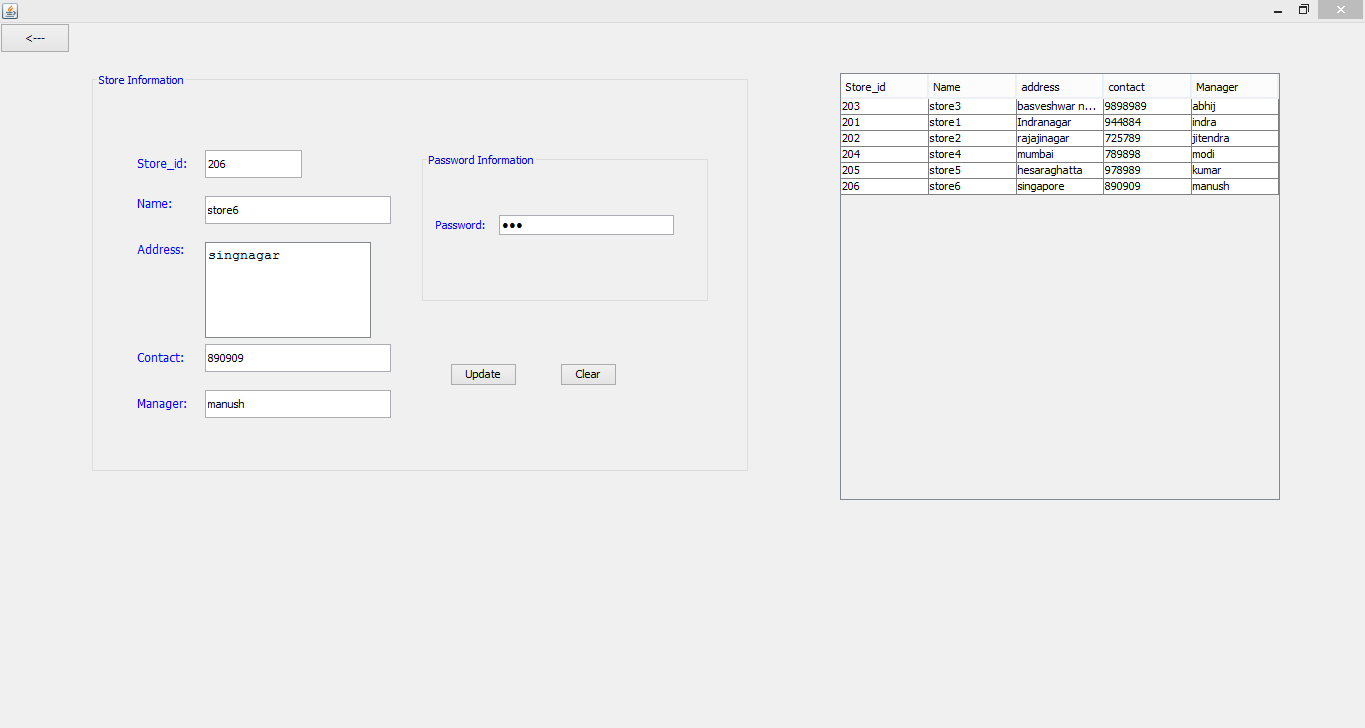


**4.8 Store Management**

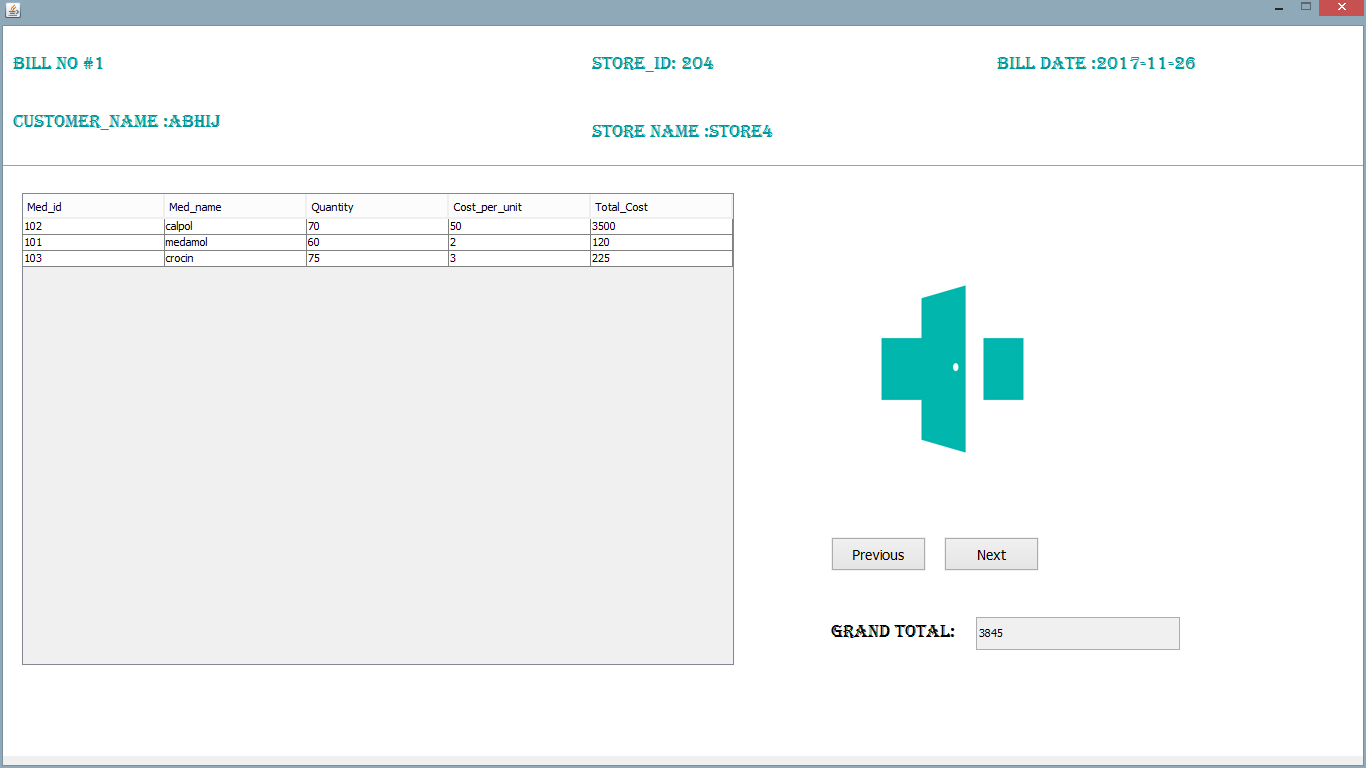
1. **Store Addition**

**5.1 Adding a new a store**

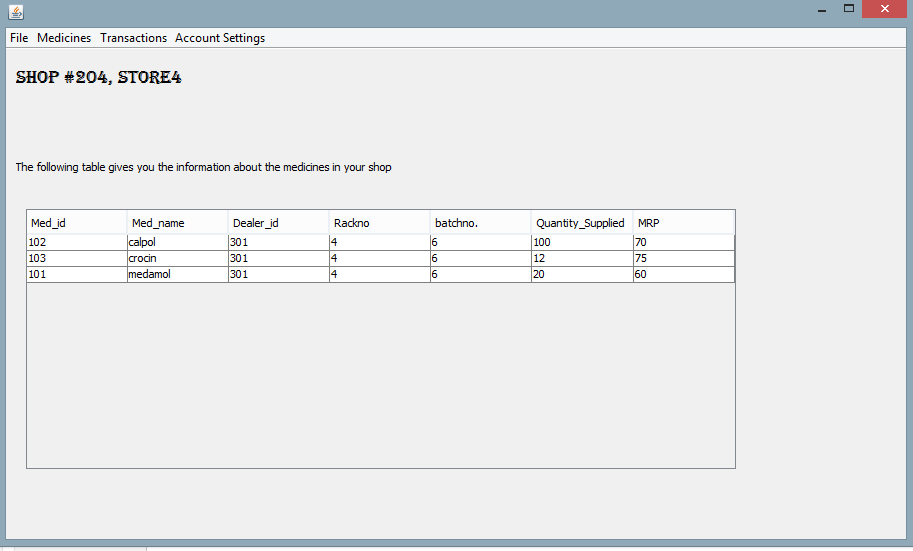


**5.1** **Updating a store**

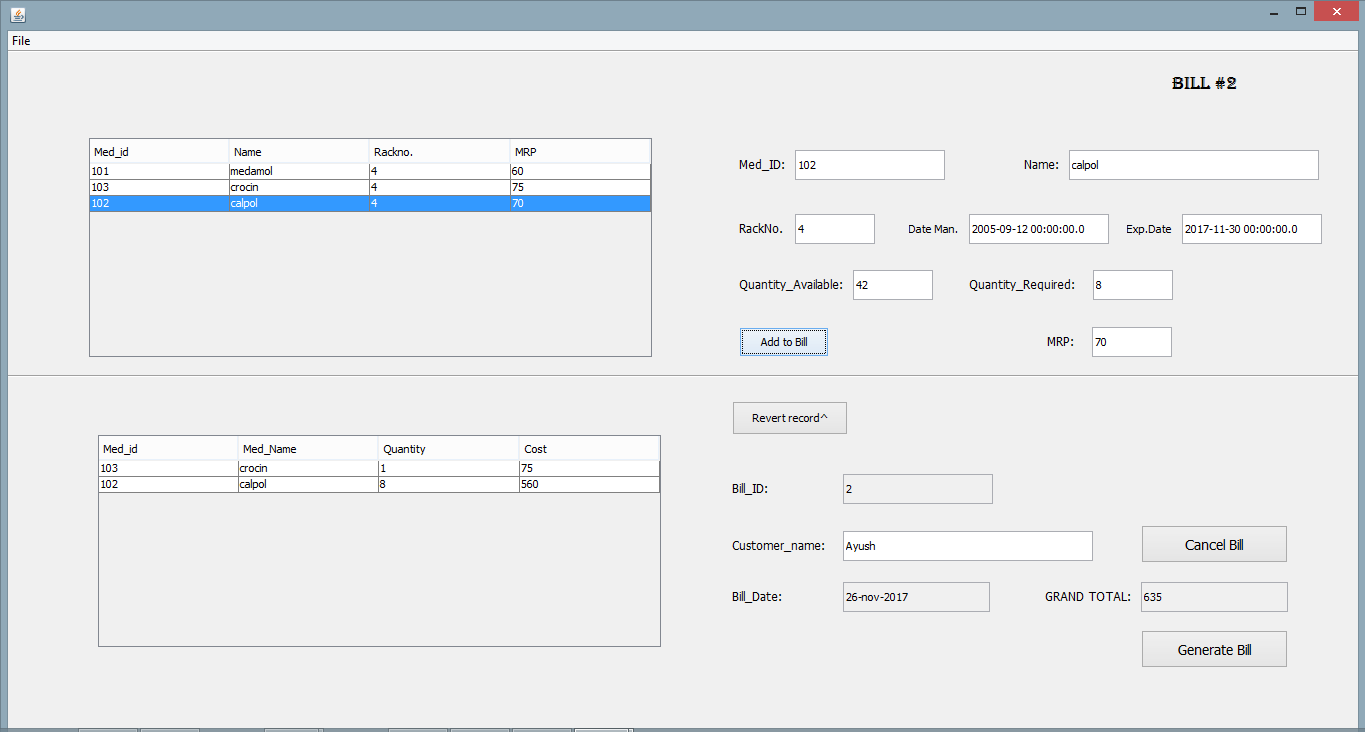
**5.2 Viewing a transaction**



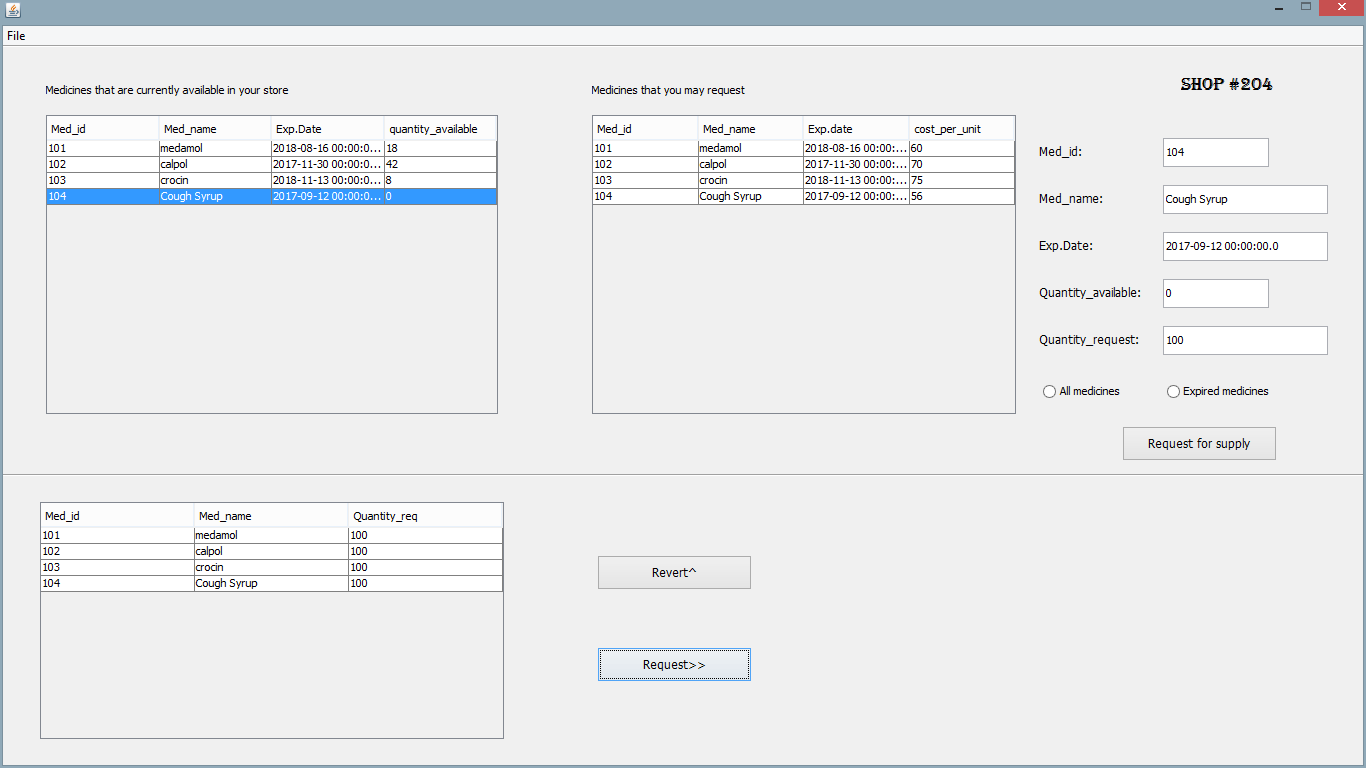
**5.3 Store Home Page**



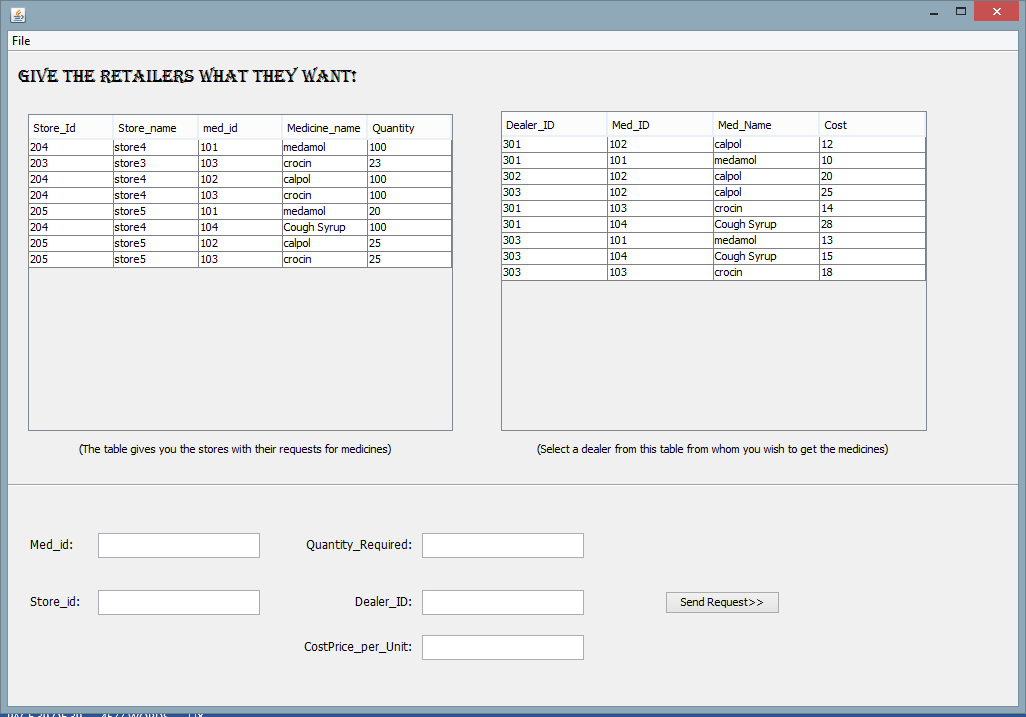
**5.4 Initiating Transactions**

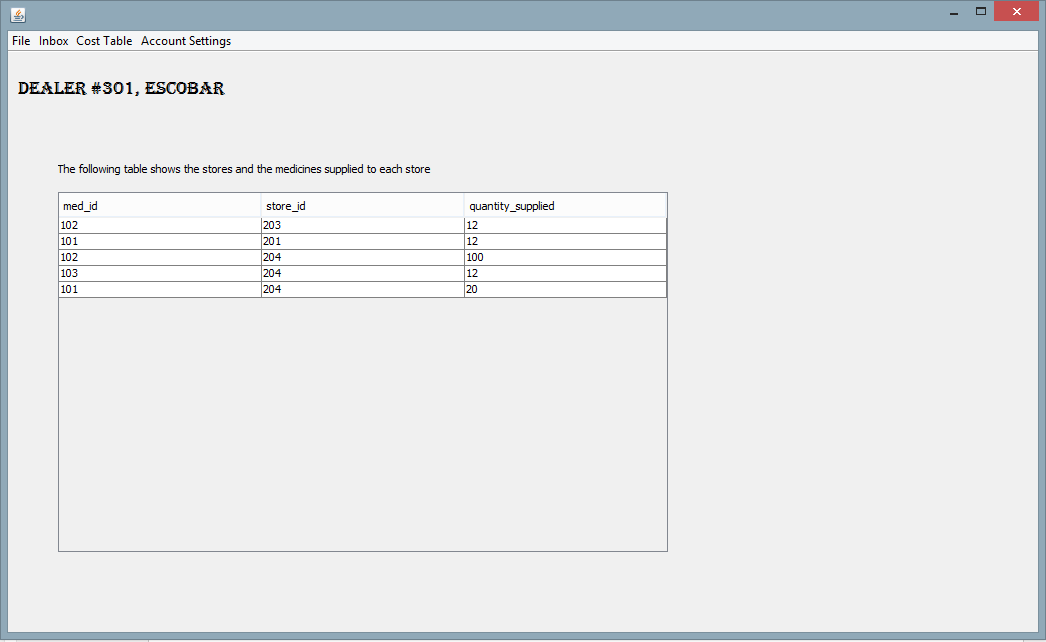


**5.5 Medicine Requests**

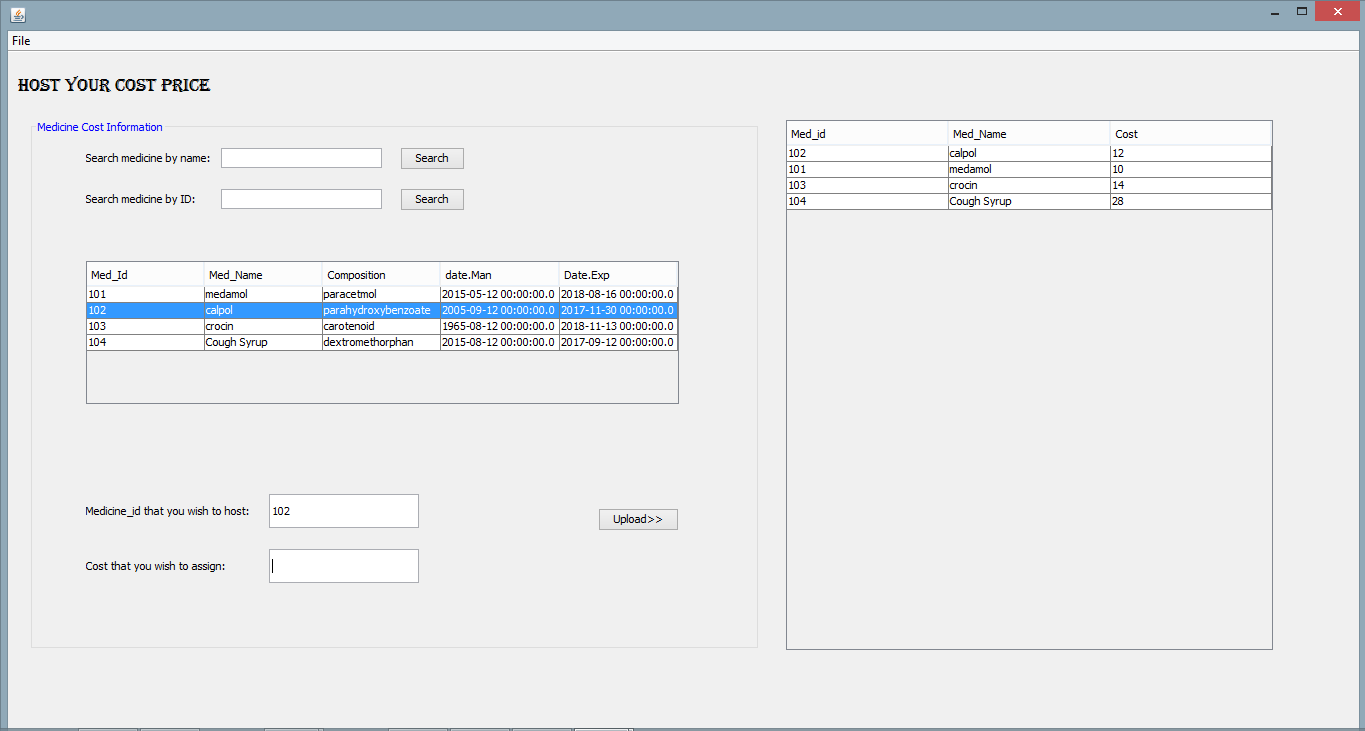


**5.6 Administrator Inbox requests**

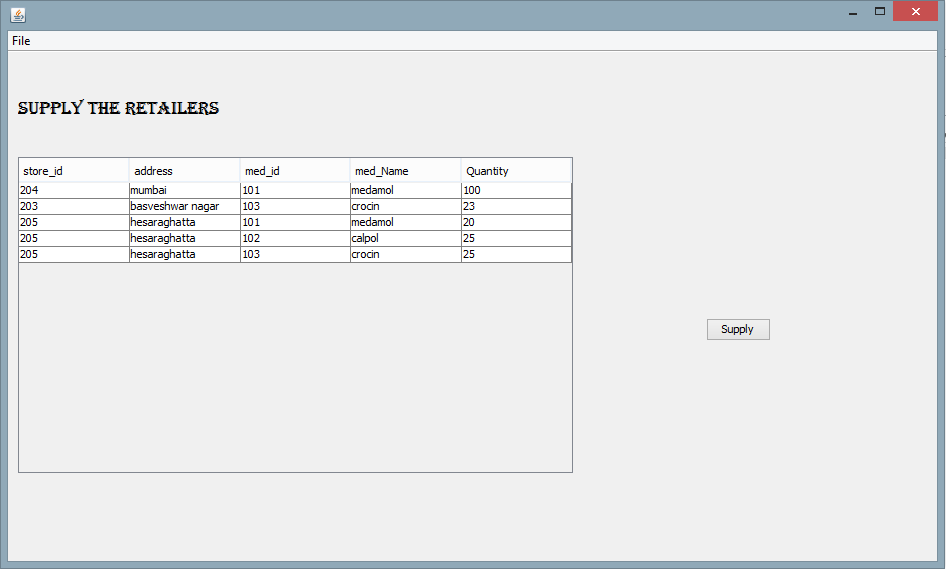


**5.7 Dealer Home Page**

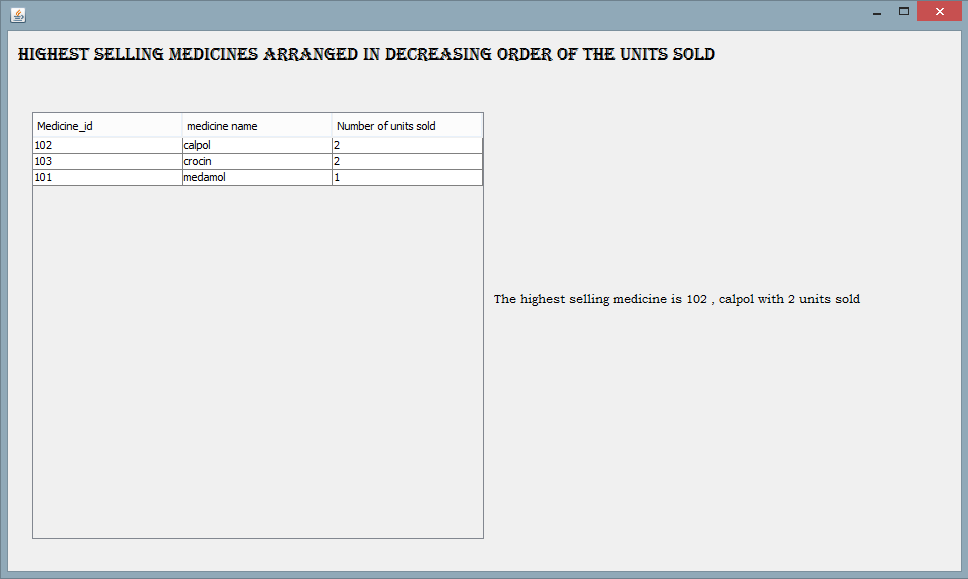
**5.8 Dealer Cost Table**

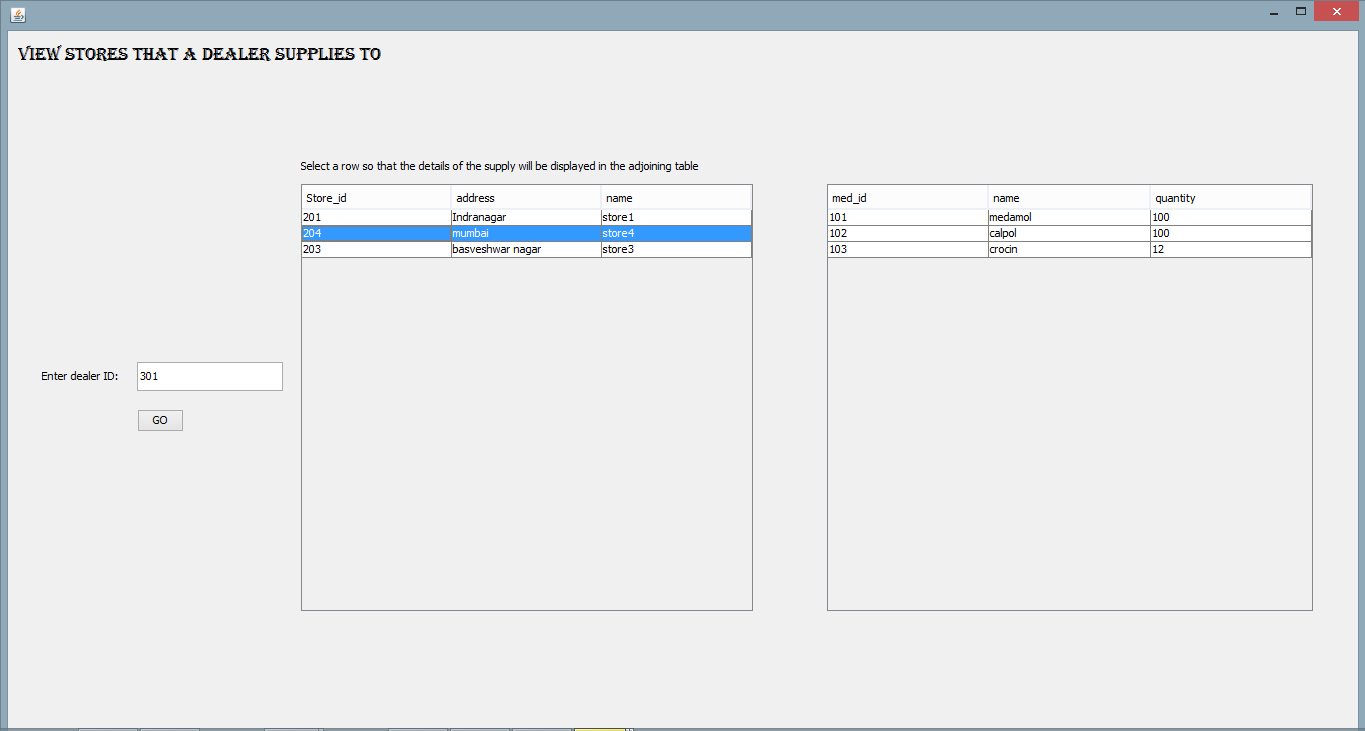


**5.9 Dealer Inbox**



**6.0 Highest Selling medicine**



**6.1 Stores that a dealer supplies to**

**CHAPTER 5**

**CONCLUSION**

The Pharmacy Management System, (its conception, organization, development and integration), has been a tremendous task and the objectives have been well accomplished. The system provides the user with a user-friendly interface. Textual data entry (for obtaining information) is minimized as much as possible by replacing it with GUI constructs. The focus lies on displaying the right information to the user from a huge pool of data. Relating the data, and retrieving the related data makes it easy to use.

**Future Works**

* This system can be enhanced by having provisions to calculate the cost price incurred to buy medicines from the dealer, and to formulate the profit data (using the transaction data produced by each store). Appropriate statistical quantities may therefore by calculated, organized and shown.
* The details of prescription can also be included- the doctor name, the illness treated etc.
* Each bill can have a code (qr or bar) and a scanner can be used to scan this to retrieve information about the transaction
* An android based mobile application can be developed which is easy to use and portable.