**PROJECT REPORT ON**

“INVENTORY DATABASE MANAGEMENT”

(FOR DMSRDE)



**DEFENCE MATERIALS & STORES RESEARCH&DEVELOPMENT**

**ESTABLISHMENT,KANPUR**

**PROJECT GUIDE: SUBMITTED BY:**

**Pranshu Awasthi**

**Mr. SANJEEV KUMAR(Sc. ‘E’) B.Tech III Year**

**1403310149**

**DEPARTMENT OF COMPUTER SCIENCE**

**Raj Kumar Goel Institute Of Technology**

**Aktu, Ghaziabad**

Preface

The innovation of computers has drastically effected organization of all kinds, be it large or small. The volume of data processed in all type of industries, medical services, financial institutes and Agencies expanding geometrically seems unimaginable to function without computers .I have been assigned the project topic “HRD DATABASE MANAGEMENT” in this Organization.

DRDO has been authorized to hold Employee Database. Database management, the repository of information is the foundation of the entire computerized information management system for an enterprise and has evolved from a specialized computer application to a central component of a modern computing environment.

Database systems are designed to manage large bodies of information. The management of data involves both the definition of structures for the storage of information and provision of mechanism for the information.

ACKNOWLEDGEMENT

I feel greatly indebted to all those who helped me during the project work. I avail this opportunity for thanking them.I sincerely express my indebtness and gratitude to **Dr N Eswara Prasad** ,D.M.S.R.D.E Kanpur & **Mr. S.D Khatri** Directorate Head DTS-I DMSRDE, for giving me an opportunity to enhance my skills and allowing me to join this esteemed organization for 4 weeks project training.

This acknowledgement would be incomplete if I fail to express deep gratitude to **Mr. Sanjeev Kumar** for his valuable guidance , patience and consummate support. Finally, I would like to thank Tec coordination section and whole staff of Administration centre D.M.S.R.D.E., Kanpur for the valuable support they rendered and for providing such a stimulating environment in which the project development was done with style and verve.

I am also thankful to my H.O.D **Dr. Anand Pandey** and in-charge of training and placement, **MS. Dhowmya Bhatt** and all faculty member of my institute for their suggestions.

**CERTIFICATE**

This is to certify that the project entitled “Searching & Sorting Using Pointers” has been developed by **“Pranshu Awasthi**”.

He has worked on the project ”Inventory Database Mangement”.He has used Java on Windows Operating System. His work is good & satisfactory.

I wish him all the best for his bright future.

**Dated:- Project In-charge**

**Mr. Sanjeev Kumar**

ORGANIZATION PROFILE

The origins of the Defence Materials & Stores Research & Development Establishment (DMSRDE) can be traced back to 1929, when it was known as the Inspectorate of General Stores in the Harness & Saddlery Factor in Cawnpore (present day Kanpur). After 1929, the establishment underwent a number of reorganizations. Finally, in 1976, the Defence Materials & Stores Research & Development Establishment was established through the amalgamation of three separate establishments.

The DMSRDE is involved in interdisciplinary research and development of materials for the military services, including various types of chemical protective clothing and equipment. Work at the DMSRDE laboratory in Kanpur includes the synthesis and development of polymers and composite materials, as well as research and development of protective clothing and equipment against hazardous and toxic chemicals. In partnership with the Defence Research and Development Establishment (DRDE), the DMSRDE has produced five types of protective systems and equipment that have been introduced into the services. These include nuclear, biological, and chemical (NBC) individual protection equipment, NBC collective protection system, NBC medical protection equipment, NBC detection equipment, and a NBC decontamination system.

The DMSRDE Kanpur laboratory interacts with the three military services, as well as a number of academic institutions, including the Indian Institute of Technology, Kanpur, and the Indian Institute of Technology, Mumbai.

INTRODUCTION

Organization requires the services of a large number of personnel. D.M.S.R.D.E involves in R&D, is a DRDO lab working in non-metallic materials, textiles and critical scientific and technological area. It has strong manpower consisting of around 200 officers and 600 staffs. These personnel occupy the various position created through the process of organizing. Each position of the organization has certain specific contribution to achieve organization objectives.

Now to provide various facilities to these employees is the responsibility of Government of India and management of the former is the duty of the organization. Organization has been authorized to hold its Employee Record.

D.M.S.R.D.E. also manage the records of offered courses, Higher Degree, Junior Research Fellowship, Senior Research Fellowship etc.

Database management, the repository of information is the foundation of the entire computerized information management system for an enterprise and has evolved from a specialized computer application to a central component of a modern computing environment.

Database systems are designed to manage large bodies of information. The management of data involves both the definition of structures for the storage of information and provision of mechanism for the information.

All the above mentioned work can be done using computer to endeavor fast and secure retrieval of data, updation and maintenance of required a complete database. For the above work, I have used most trusted Database ‘ORACLE 10g’ and JSP ,which is capable of showing various kinds of forms and reports important in making such kind of project.

The overall Project maintains tables with suitable and secure relation between them . This can be understand by the following E-R Diagram and table definitions.

**TOOLS AND PLATFORM :-**

Designed by using …

Front end: JDK 1.8, JSP, Servlet

Back end: SQL

Operating System: MS Windows’s 7/8/10

Server: Apache Tomcat 7.x

Some Attractive features of project

**\*** User can perform all database changes in application.

**\*** More flexible & reliable

**\*** Plateform independent application

**\*** Can run on any version of java environment

**\*** 3-tyre Architecture

# SQL DATABASE

SQL stands for Structured Query Language. SQL is used to communicate with a database. According to ANSI (American National Standards Institute), it is the standard language for relational database management systems. SQL statements are used to perform tasks such as update data on a database, or retrieve data from a database. Some common relational database management systems that use SQL are: Oracle, Sybase, Microsoft SQL Server, Access, Ingres, etc. Although most database systems use SQL, most of them also have their own additional proprietary extensions that are usually only used on their system. However, the standard SQL commands such as "Select", "Insert", "Update", "Delete", "Create", and "Drop" can be used to accomplish almost everything that one needs to do with a database.

The library cache stores shared SQL, caching the parse tree and the execution plan for every unique SQL statement.

If multiple applications issue the same SQL statement, each application can access the shared SQL area. This reduces the amount of memory needed and reduces the processing-time used for parsing and execution planning.

The data dictionary comprises a set of tables and views that map the structure of the database.

The data dictionary contains information such as:

\* user information, such as user privileges

\* integrity constraints defined for tables in the database

\* names and datatypes of all columns in database tables

\* information on space allocated and used for schema objects

# Configure JDBC Connection to SQL Server

* Right-click on your Java project and Select Properties
* Choose Java Build Path and Select Third tab
* Add External JAR to add JAR from any folder on your machine. It will open file chooser to choose the JAR file from the file system and add that into CLASSPATH.

Simple Code For Connection TO Database-

**import** java.sql.\*;

**class** MysqlCon{

**public** **static** **void** main(String args[]){

**try**{

Class.forName("com.mysql.jdbc.Driver");

Connection con=DriverManager.getConnection(

"jdbc:mysql://localhost:3306/sonoo","root","root");

//here sonoo is database name, root is username and password

Statement stmt=con.createStatement();

esultSet rs=stmt.executeQuery("select \* from emp");

**while**(rs.next())

System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));

con.close();

**catch**(Exception e){ System.out.println(e);}

# }

# List the advantages of JDBC

1. **Provide Existing Enterprise Data**  
Businesses can continue to use their installed databases and access information even if it is stored on different database management systems.  
  
2. **Simplified Enterprise Development**  
The combination of the Java API and the JDBC API makes application development easy and cost effective.  
  
3. **Zero Configuration for Network Computers**  
No configuration is required on the client side centralizes software maintenance. Driver is written in the Java, so all the information needed to make a connection is completely defined by the JDBC URL or by a DataSource object. DataSource object is registered with a Java Naming and Directory Interface (JNDI) naming service.  
  
4. **Full Access to Metadata**  
The underlying facilities and capabilities of a specific database connection need to be understood. The JDBC API provides metadata access that enables the development of sophisticated applications.   
  
5. **No Installation**  
A pure JDBC technology-based driver does not require special installation.  
  
6. **Database Connection Identified by URL**  
The JDBC API includes a way to identify and connect to a data source, using a DataSource object. This makes code even more portable and easier to maintain.

# 

JAVA SERVER PAGES

**JavaServer Pages** (**JSP**) is a Java technology that allows software developers to create dynamically generated web pages, with HTML, XML, or other document types, in response to a Web client request. The technology allows Java code and certain pre-defined actions to be embedded into static content.

The JSP syntax adds additional XML-like tags, called JSP actions, to be used to invoke built-in functionality. Additionally, the technology allows for the creation of JSP tag libraries that act as extensions to the standard HTML or XML tags. Tag libraries provide a platform independent way of extending the capabilities of a Web server

## Servlets

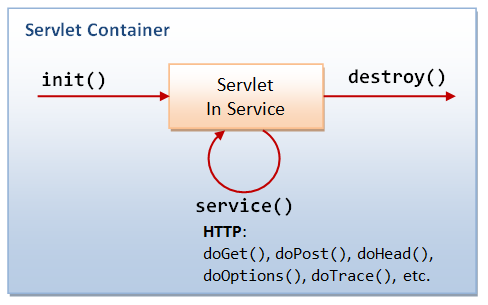
Architecturally, JSP may be viewed as a high-level abstraction of Java servlets. Both servlets and JSPs were originally developed at Sun Microsystems. Starting with version 1.2 of the JSP specification, JavaServer Pages have been developed under the Java Community Process. JSR 53 defines both the JSP 1.2 and Servlet 2.3 specifications and JSR 152 defines the JSP 2.0 specification. As of May 2006 the JSP 2.1 specification has been released under JSR 245 as part of Java EE 5.

JSPs are compiled into servlets by a JSP compiler. The compiler either generates a servlet in Java code that is then compiled by the Java compiler, or it may compile the servlet to byte code which is directly executable. JSPs can also be interpreted on-the-fly, reducing the time taken to reload changes.

# Servlets Life Cycle-

A servlet life cycle can be defined as the entire process from its creation till the destruction. The following are the paths followed by a servlet.

* The servlet is initialized by calling the **init()** method.
* The servlet calls **service()** method to process a client's request.
* The servlet is terminated by calling the **destroy()** method.
* Finally, servlet is garbage collected by the garbage collector of the JVM.

[](http://4.bp.blogspot.com/-dFe1L1RJEK0/UxnO6SKSedI/AAAAAAAAAtE/xSbkCQ1hQi4/s1600/Servlet_LifeCycle.png)

## Simple Code For Servlet

**import** javax.servlet.http.\*;

**import** javax.servlet.\*;

**import** java.io.\*;

**public** **class** DemoServlet **extends** HttpServlet{

**public** **void** doGet(HttpServletRequest req,HttpServletResponse res)

**throws** ServletException,IOException

{

res.setContentType("text/html");

PrintWriter pw=res.getWriter();

pw.println("<html><body>");

pw.println("Welcome to servlet");

pw.println("</body></html>");

pw.close();

}}

# Servlets Architecture

The following diagram shows the position of Servlets in a Web Application.

## Servlets Architecture

# The init() Method

The init method is called only once. It is called only when the servlet is created, and not called for any user requests afterwards. So, it is used for one-time initializations, just as with the init method of applets.

The servlet is normally created when a user first invokes a URL corresponding to the servlet, but you can also specify that the servlet be loaded when the server is first started.

.

The init method definition looks like this −

public void init() throws ServletException {

// Initialization code...

}

# The service() Method

The service() method is the main method to perform the actual task. The servlet container (i.e. web server) calls the service() method to handle requests coming from the client( browsers) and to write the formatted response back to the client.

Each time the server receives a request for a servlet, the server spawns a new thread and calls service. The service() method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate.

Here is the signature of this method −

public void service(ServletRequest request, ServletResponse response)

throws ServletException, IOException {

}

The service () method is called by the container and service method invokes doGe, doPost, doPut, doDelete, etc. methods as appropriate. So you have nothing to do with service() method but you override either doGet() or doPost() depending on what type of request you receive from the client.

The doGet() and doPost() are most frequently used methods with in each service request. Here is the signature of these two methods.

# The doGet() Method

A GET request results from a normal request for a URL or from an HTML form that has no METHOD specified and it should be handled by doGet() method.

public void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

// Servlet code

}

# The doPost() Method

A POST request results from an HTML form that specifically lists POST as the METHOD and it should be handled by doPost() method.

public void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

// Servlet code

}

# The destroy() Method

The destroy() method is called only once at the end of the life cycle of a servlet. This method gives your servlet a chance to close database connections, halt background threads, write cookie lists or hit counts to disk, and perform other such cleanup activities.

After the destroy() method is called, the servlet object is marked for garbage collection. The destroy method definition looks like this −

public void destroy() {

// Finalization code...

}

# Model-view-controller paradigm (MVC)

Sun recommends that the Model-view-controller pattern be used with the JSP files in order to split the presentation from request processing and computer data storage. Either regular servlets or separate JSP files are used to process the request. After the request processing has finished, control is passed to a JSP used only for creating the output. There are several platforms based on Model-view-controller pattern for web tiers (such as Barracuda, Apache Struts, Stripes or Spring MVC framework).

## Coding JSP

JSP pages embed tags within a HTML or XML document that are evaluated by the compiler. This can be done through the use of scripting tags similar to those used in PHP or ASP.NET, or by importing a JSP Tag Library.

### JSP Directives

JSP directives are added at the top of a JSP page. These directives control how the JSP compiler generates the servlet. The following directives are available:

**include**

The include directive informs the JSP compiler to include a complete file into the current file. It is as if the contents of the included file were pasted directly into the original file. This functionality is similar to the one provided by the C preprocessor. Included files generally have the extension "jspf" (for JSP **Fragment**):

<%@ include file="somefile.jspf" %>

**page**

The page directive has several attributes:

**import**

Results in a Java import statement being inserted into the resulting file.

**contentType**

Specifies the content that is generated. This should be used if HTML is not used or if the character set is not the default character set.

**errorPage**

Indicates the address of the page that should be shown if an exception occurs while processing the HTTP request.

**isErrorPage**

If set to true, it indicates that this is the error page. Default value is *false*.

**isThreadSafe**

A boolean indicating whether the resulting servlet is [thread](http://en.wikipedia.org/wiki/Thread_(computer_science)) safe.

**autoFlush**

To autoflush the contents. A value of true, the default, indicates that the buffer should be flushed when it is full. A value of false, rarely used, indicates that an exception should be thrown when the buffer overflows. A value of false is illegal when also using buffer="none".

**session**

To maintain session. A value of true (the default) indicates that the predefined variable session (of type HttpSession) should be bound to the existing session if one exists, otherwise a new session should be created and bound to it. A value of false indicates that no sessions will be used, and attempts to access the variable session will result in errors at the time the JSP page is translated into a servlet.

**buffer**

To set Buffer Size. The default is 8k and it is advisable that you increase it.

**isELIgnored**

Defines whether EL expressions are ignored when the JSP is translated.

**language**

Defines the scripting language used in scriptlets, expressions and declarations. Right now, the only possible value is "java".

**extends**

Defines the superclass of the class this JSP will become. You won't use this unless you REALLY know what you're doing - it overrides the class hierarchy provided by the Container.

**info**

Defines a String that gets put into the translated page, just so that you can get it using the generated servlet's inherited getServletInfo() method.

**pageEncoding**

Defines the character encoding for the JSP. The default is "ISO-8859-1"(unless the contentType attribute already defines a character encoding, or the page uses XML document syntax).

<%@ page import="java.util.\*" %> <%-- example import --%>

<%@ page contentType="text/html" %> <%-- example contentType --%>

<%@ page isErrorPage="false" %> <%-- example for non error page --%>

<%@ page isThreadSafe="true" %> <%-- example for a thread safe JSP --%>

<%@ page session="true" %> <%-- example for using session binding --%>

<%@ page autoFlush="true" %> <%-- example for setting autoFlush --%>

<%@ page buffer="20kb" %> <%-- example for setting Buffer Size --%>

**Note:** Only the "import" page directive can be used multiple times in the same JSP.

**taglib**

The taglib directive indicates that a JSP tag library is to be used. The directive requires that a prefix be specified (much like a namespace in C++) and the URI for the tag library description.

<%@ taglib prefix="myprefix" uri="taglib/mytag.tld" %>

### Implicit Objects

The JSP container exposes a number of implicit objects that can be used by the programmer:

out

The JspWriter used to write the data to the response stream.

page

The servlet itself.

pageContext

A PageContext instance that contains data associated with the whole page. A given HTML page may be passed among multiple JSPs.

request

The HttpServletRequest object that provides HTTP request information.

response

The HttpServletResponse object that can be used to send data back to the client.

session

The HttpSession object that can be used to track information about a user from one request to another.

config

Provides servlet configuration data.

application

Data shared by all JSPs and servlets in the application.

exception

Exceptions not caught by application code.

### Scripting Elements

There are *three* basic kinds of scripting elements that allow java code to be inserted directly into the servlet.

* A ***declaration*** tag places a variable definition inside the body of the java servlet class. Static data members may be defined as well. Also inner classes should be defined here.

<%! int serverInstanceVariable = 1; %>

Declaration tags also allow methods to be defined.

<%!

/\*\*

\* Converts the Object into a string or if

\* the Object is null, it returns the empty string.

\*/

public String toStringOrBlank( Object obj ){

if(obj != null){

return obj.toString();

}

return "";

}

%>

* A ***scriptlet*** tag places all of the statements contained within it, inside the \_jspService() method of the java servlet class.

<% int localStackBasedVariable = 1;

out.println(localStackBasedVariable); %>

* An ***expression*** tag places an expression to be evaluated inside the java servlet class. Expressions should not be terminated with a semi-colon .

<%= "expanded inline data " + 1 %>

### 

### JSP actions

JSP actions are XML tags that invoke built-in web server functionality. They are executed at runtime. Some are standard and some are custom (which are developed by Java developers). The following list contains the standard ones:

jsp:include  .

jsp:param

jsp:forward

jsp:plugin

jsp:fallback

jsp:getProperty

jsp:forward

APACHE TOMCAT

**Apache Tomcat** is a servlet container developed by the Apache Software Foundation (ASF). Tomcat implements the Java Servlet and the JavaServer Pages (JSP) specifications from Sun Microsystems, and provides a "pure Java" HTTP web server environment for Java code to run.

Tomcat should not be confused with the Apache web server, which is a C implementation of an HTTP web server; these two web servers are not bundled together. Apache Tomcat includes tools for configuration and management, but can also be configured by editing XML configuration files

## Development status

Members of the **ASF** and independent volunteers develop and maintain Tomcat. Users have free access to the source code and to the binary form of Tomcat under the Apache License. The initial Tomcat release appeared with versions 3.0.x (previous releases were Sun internal releases, and were not publicly released). Tomcat 6.0.20 is the latest production quality release of the 6.0.x trunk (the branch for the 2.5 servlet specification), as of 2009.

**Product features**

An overview of the different versions can be found on the Apache website.

**Tomcat 4.x**

* Released 2001
* implements the Servlet 2.3 and JSP 1.2 specifications
* servlet container redesigned as Catalina
* JSP engine redesigned as Jasper
* Coyote HTTP connector
* Java Management Extensions (JMX), JSP and Struts-based administrations

**Tomcat 5.x**

* implements the Servlet 2.4 and JSP 2.0 specifications
* reduced garbage collection, improved performance and scalability
* native Windows and Unix wrappers for platform integration
* faster JSP parsing

**Tomcat 6.x**

* implements the Servlet 2.5 and JSP 2.1 specifications
* support for Unified Expression Language 2.1
* designed to run on Java SE 5.0 and later
* support for Comet through the CometProcessor interface
* is not packaged with an admin console as in past releases

# Tomcat integration in Eclipse

* From the Eclipse main menu choose File > New > Other...
* Select Server > Server.
* Click Next. Figure: Define a New Server.
* Select Tomcat vx.x Server.
* Click Next. Figure: Choose Tomcat version.
* Browse to the folder of your Tomcat installation. ...
* Select Finish.

**Application Snapshots**

**** main page

 login page

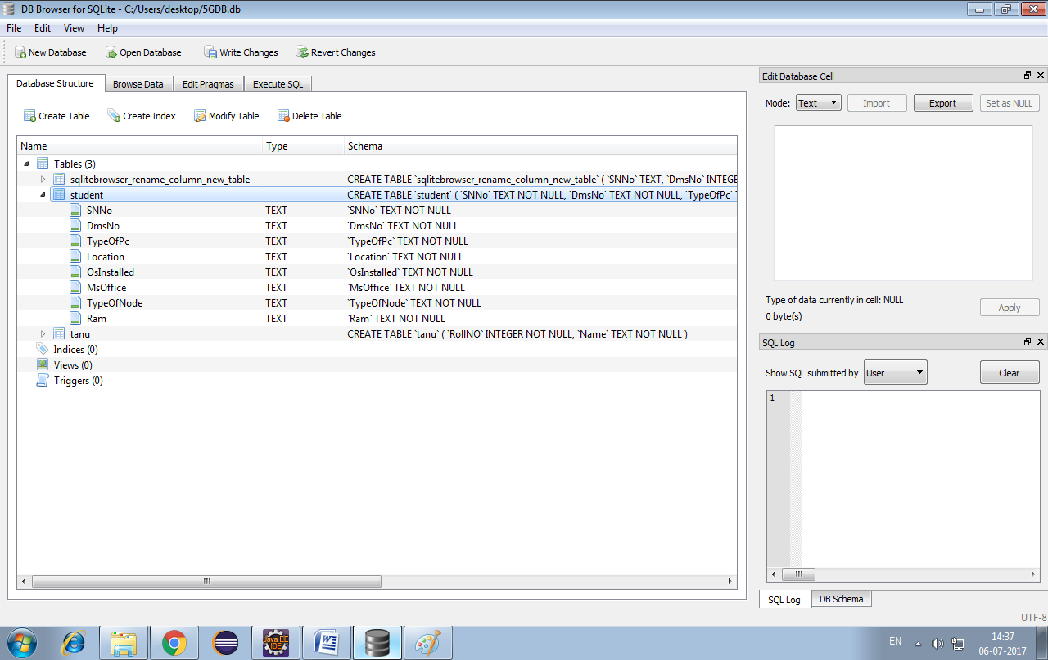
# Add Inventory Page

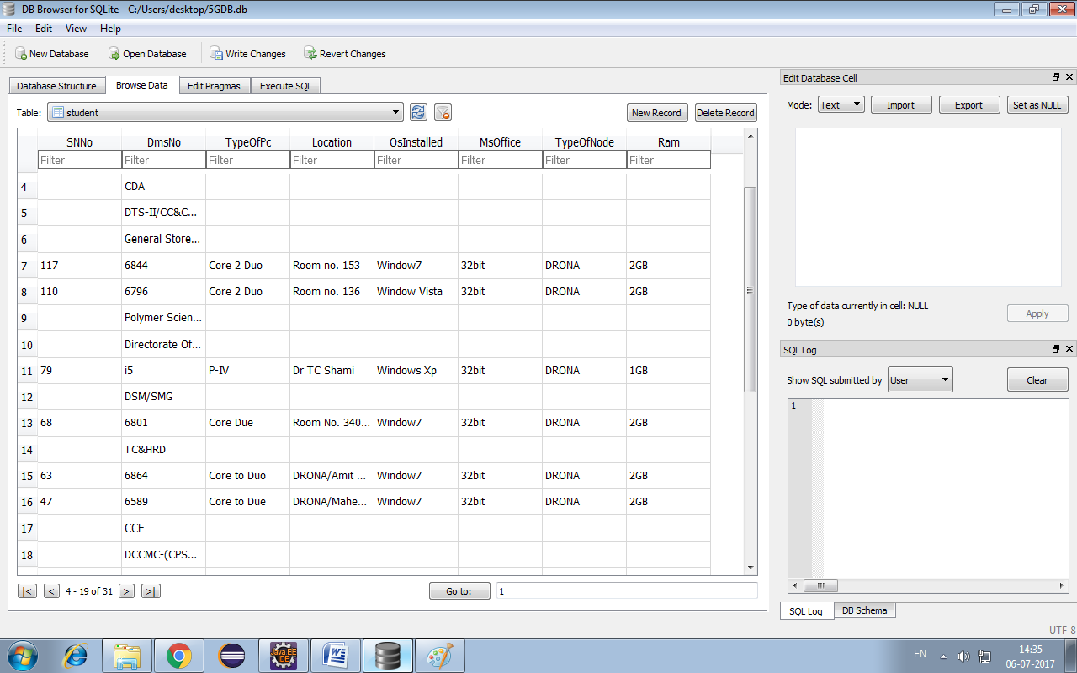


# View List

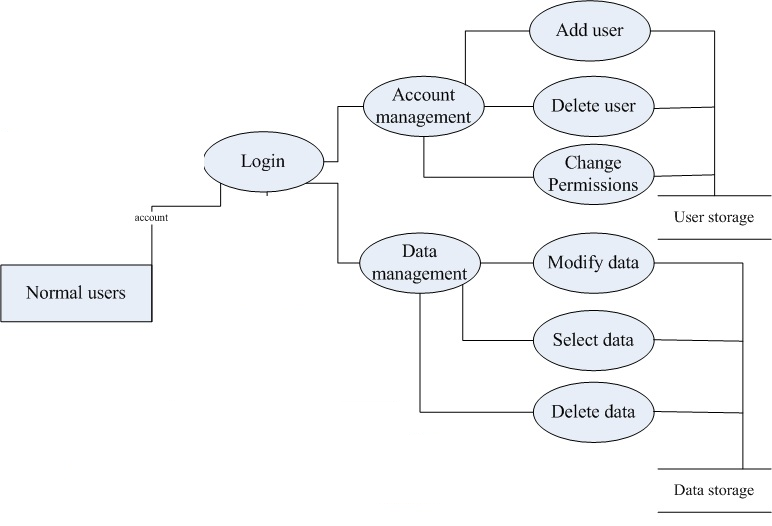
****

**DataBase SQl Tables-**

****

****

**DFD**

****

**Conclusion**

##### The main objective of the project is to develop a Inventory database Mangement. I had taken a wide range of literature review in order to achieve all the tasks, where I came to know about some of the products that are existing in the market. I made a detailed research in that path to cover the loop holes that existing systems are facing and to eradicate them in our application. In the process of research I came to know about the latest technologies and different algorithms.

**References**

* [**www.google.com**](http://www.google.com)
* [**www.javatpoint.com**](http://www.javatpoint.com)
* [**www.w3schools.com**](http://www.w3schools.com)
* [**www.tutorialspoint.com**](http://www.tutorialspoint.com)