

Copyright is owned by the Author of the thesis. Permission is given for a copy to be downloaded by an individual for the purpose of research and private study only. The thesis may not be reproduced elsewhere without the permission of the Author.

A Database with Enterprise Application for Mining Astronomical Data Obtained by MOA

By

Huawei Xu

A thesis submitted in partial fulfillment of the
requirements for the degree of the Master of
Information Science in Computer Science

Massey University at Albany,
Auckland, New Zealand
cyxhw@hotmail.com
February 2007

Abstract

The MOA (Microlensing Observations in Astrophysics) Project is one of a new generation of modern astronomy endeavours that generates huge volumes of data. These have enormous scientific data mining potential. However, it is common for astronomers to deal with millions and even billions of records. The challenge of how to manage these large data sets is an important case for researchers. A good database management system is vital for the research. With the modern observation equipments used, MOA suffers from the growing volume of the data and a database management solution is needed. This study analyzed the modern technology for database and enterprise application. After analysing the data mining requirements of MOA, a prototype data management system based on MVC pattern was developed. Furthermore, the application supports sharing MOA findings and scientific data on the Internet. It was tested on a 7GB subset of achieved MOA data set. After testing, it was found that the application could query data in an efficient time and support data mining.

Table of Contents:

1.0 INTRODUCTION/ MOA PROJECT	4
1.1 MOA PROJECT AND EQUIPMENTS.....	5
1.2 DESCRIPTION OF MOA DATA.....	9
1.3 CURRENT REPRESENTATION OF MOA DATA	11
1.4 OVERVIEW OF THIS PROJECT	12
2.0 BACKGROUND ON DATA MANAGEMENT	13
2.1 INTRODUCTION OF DATABASE.....	13
2.1.1 <i>Manual Filing system</i>	13
2.1.2 <i>Traditional File-based systems</i>	14
2.1.3 <i>Database approach</i>	14
2.2 JAVA APPLICATION.....	17
2.2.1 <i>Java technology</i>	17
2.2.2 <i>Java Enterprise Application</i>	17
2.2.3 <i>Java Enterprise Application Architecture</i>	17
2.2.4 <i>The advantages of J2EE</i>	19
2.2.5 <i>Design Pattern --MVC in J2EE</i>	20
3.0 REVIEWS EXISTING ASTRONOMY DATABASE TECHNOLOGIES AND SOLUTIONS	23
3.1 SOME EXISTING ASTRONOMICAL PROJECTS INVOLVING LARGE VOLUMES OF DATA	23
3.2 POSSIBLE DATABASE SOLUTION IN THE MARKET FOR ASTRONOMY DATABASE .	26
4.0 PROJECT DEVELOPMENT	28
4.1 REQUIREMENTS COLLECTIONS AND ANALYSIS	28
4.1.1 <i>MOA database</i>	28
4.1.2 <i>The web application</i>	31
4.2 APPLICATION ARCHITECTURAL DESIGN	32
4.3 DATABASE DESIGN	33
4.3.1 <i>Conceptual database design</i>	33
4.3.2 <i>Logical database design</i>	37
4.3.3 <i>Physical database design</i>	49

4.4 DATA INGESTION APPLICATION DESIGN	52
4.5 ENTERPRISE APPLICATION DESIGN	60
4.5.1 <i>Application architecture design</i>	60
4.5.2 <i>Servlet Controller design (Controller layer)</i>	62
4.5.3 <i>Presentation logic design (View layer)</i>	63
4.5.4 <i>Business logic design (Java Bean Model layer)</i>	69
4.5.5 <i>Data access design</i>	71
4.5.6 <i>An example of passing messages among different layers in MVC pattern.</i> 72	
4.5.7 <i>Techniques used in the application</i>	76
4.5.8 <i>Maintaining The Application</i>	85
5.0 PROJECT DEPLOYMENT	87
5.1 DATABASE DEPLOYMENT.....	87
5.1.1 <i>MOA database creating</i>	87
5.1.2 <i>Data Loading</i>	87
5.2 ENTERPRISE APPLICATION DEPLOYMENT	88
6.0 PROJECT PERFORMANCE	90
6.1 DATABASE PERFORMANCE ANALYSIS	90
6.2 DATABASE SUPPORTING SCIENTIFIC DATA MINING.....	91
6.3 ENTERPRISE APPLICATION PERFORMANCE	94
7.0 CONCLUSION	99
8.0 FUTURE WORK.....	100
REFERENCES	101
APPENDIX A -- SQL FOR CREATING MOA DATABASE.....	103

1.0 Introduction/ MOA project

On a clear night, light has travelled through space for about millions of years and then reaches the earth. It quickly disappears but its trace as photons is captured by astronomical telescope and is recorded as the digital images. Then astronomers digitize the images and start their research on the mystical universe.

The combination of modern computer technology, wide-field CCD detectors and telescope technology have opened up new opportunities for astronomy research. As a result, astronomical research groups have used these advanced technologies in their study. For example, as the most capable Infrared imaging survey instrument, WFCAM is used by the UK Infrared Telescope on Mauna Kea (The Royal Observatory Edinburgh, 2005). This instrument is built by a cryogenic camera with four state-of-the-art detectors (four Rockwell 2048x2048 detectors), associated electronics and computing. Furthermore, its large optics, including a new f/9 secondary and a complete auto guiding system, supports to take perfect photos in a short time and its software is able to process the real-time pipeline data in time.

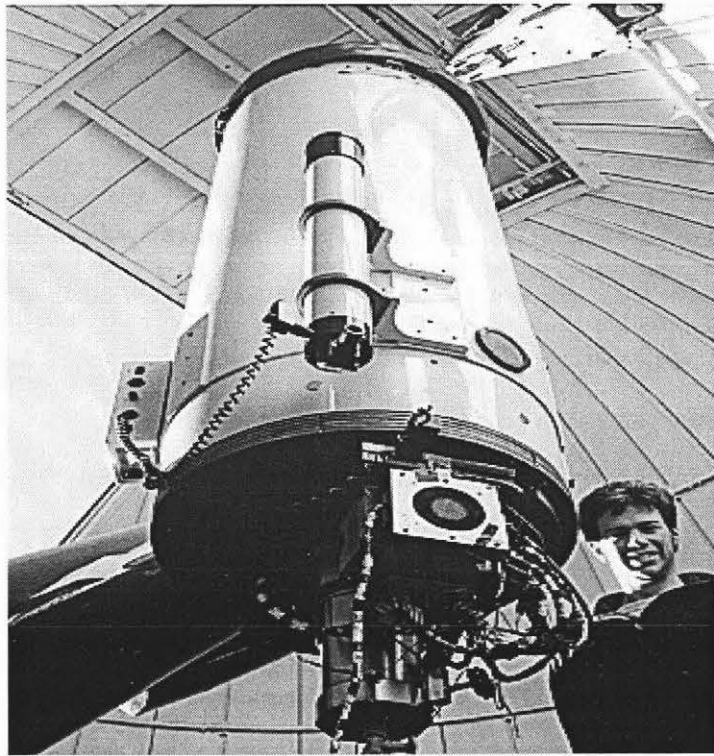
At the same time, in order to improve the survey quality, scientists have been built more and more professional telescopes. For instance, SkyMapper telescope is dedicated to scanning the night time sky. It is a 1.35m survey telescope with an 8-sq degree field of view. And it has an integrated 16kx16k mosaic CCD and will be on operation in 2007. SkyMapper telescope will be able to scan the night time skies quicker and deeper than before (RSAA, 2006). As a space-based telescope, the famous Hubble Space Telescope (HST) is 2.4m reflecting telescope in orbit around the Earth. HST was designed as a long term observatory and the instruments include three cameras, two spectrographs, and fine guidance sensors (Space Telescope Science Institute, 2006). Because of HST's location above the Earth's atmosphere, this science instruments can produce high resolution images of astronomical objects than the ground-based telescope. In addition, advanced ground-based telescopes also are commonly used for observation as well. Large Synoptic Survey Telescope (LSST) is a planned wide-field "survey" reflecting telescope and it will be used in 2013. The LSST has an unbelievable size-8m and a wide field view for exposure. Its 3.2 billion-pixel camera can take a 15-second exposure in every 20 seconds. This camera

is expected to take over 200,000 pictures (1.28 pet bytes uncompressed) per year (LSST, 2006). In contrast, Visible and Infrared Survey Telescope (VISTA) will be available early than LSST; it will be in field in 2007. It will be a 4m class wide field survey telescope with a near infrared camera containing 67 million 0.34 arcsec pixels and available broad band filter. VISTA will produce about 315 GB on a typical night (Emerson, 2006).

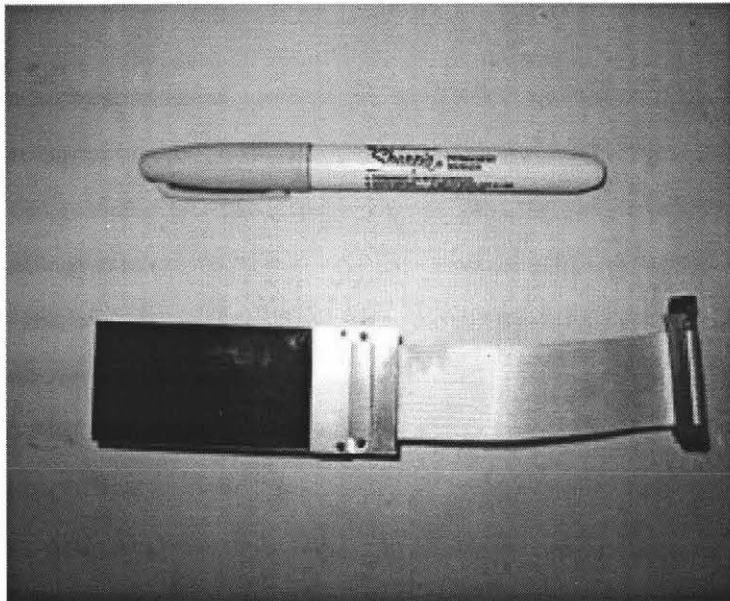
However, with the use of new modern equipment, the volume of data is large and is being accumulated at higher and higher rates. Managing and effectively data mining the enormous telescope output data is the most technical and difficult part of the projects. Therefore, astronomers need to discover a suitable database management system for them.

1.1 MOA Project and Equipments

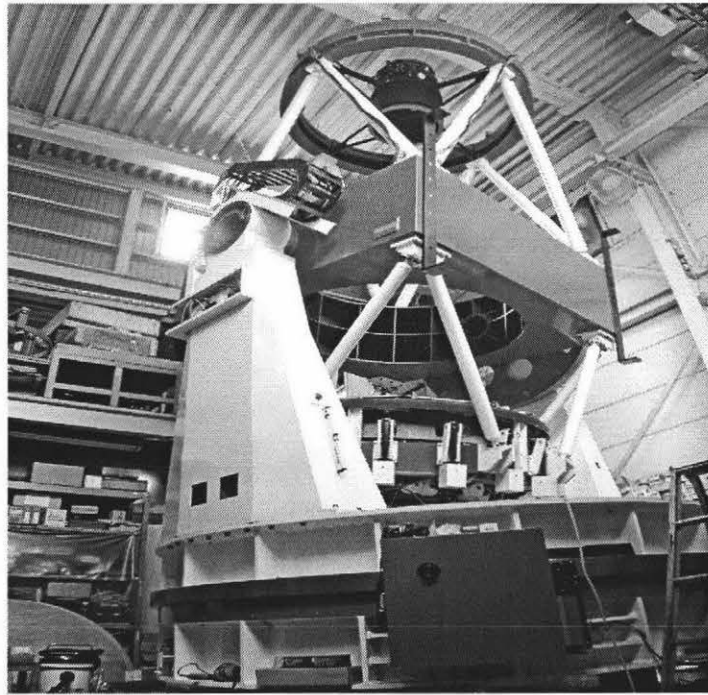
Microlensing Observations in Astrophysics (MOA) is a joint Japanese/New Zealand collaboration designed to perform large-scale astronomical photometry to detecting interesting astrophysical phenomena. MOA is one of a new generation of astronomical research projects. It started in 1995; it makes “observations and measurements on dark matter, extra-solar planets and stellar atmospheres using the gravitational microlensing technique” (MOA, 2006). Between 1998 and 2005, MOA used a 0.6-metre Boller and Chivens telescope for the observation. Picture 1 shows the 0.6m telescope with mosaic CCD camera called MOACam2 and it has 4096x6144-pixels. MOACam2 has three 2048x4069-pixel site CCD chips and the Picture 2 shows one of them. In 2002, MOA received a new 1.8-metre-diameter telescope from the Ministry of Education, Culture, Science & Technology of Japan. It is the largest telescope in New Zealand at the moment; Picture 3 shows it. During 2005, the new MOA-II telescope (Picture 4) equipped with new mosaic CCD camera called MOACam3 (Picture 5) launched was commissioned. The MOACam3 is an 80Mpixels CCD camera, which has ten CCD chips, and each chip is 2048x4096 pixels.



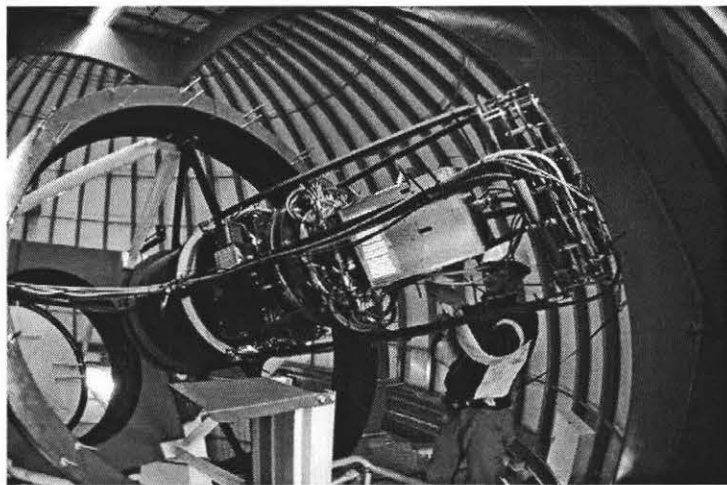
Picture 1: The 0.6 m telescope with MOACam2 attached



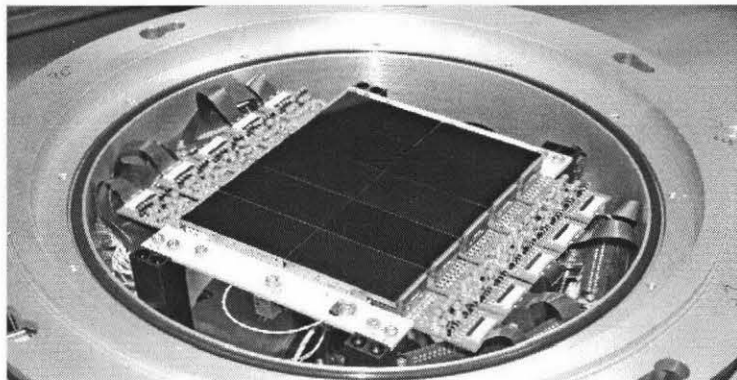
Picture 2: One of CCD chip used in MOACam2



Picture 3: The new 1.8m MOA-II telescope



Picture 4: MOACam3 attached to MOA-II telescope



Picture 5: MOACam3 with 10 CCD chips

Obtained Data

Table 1 shows the volume of data MOA obtained among different phases. From Table 1 we can see, there were about 500 GB data obtained by MOA-I telescope with MOACam1 between 1995 and 1997. In contrast, for the observations that were carried out by the new MOACam2 from 1997 to 2004, the accumulated data increased to 3TB. Since 2005 the data volume has increased dramatically. When the MOA-II telescope with new camera was launched, MOA obtains more than 3TB data each year. With the growth of the data size, how to manage these data is a challenge for MOA.

Table 1: Timeline of MOA data

Time	Telescope	CCD Camera	Number of CCD	Dimensions of each CCD	Data produced
1995-1997	0.6m	MOACam1	9	1000 x 1018	About 500GB
1997-2004	0.6m	MOACam2	3	2048 x 4096	About 3 TB
2005-now	1.8m	MOACam3	10	2048 x 4096	3 TB per year

At the moment, MOA uses a set of Python scripts to analyze the data. Roughly, they use three steps to do the research. Firstly, MOA members transform imaging data into measurement that records position, fluxes and other information of the observed objects. During the transformation, the data sizes are reduced and the useful information is retained. Secondly, MOA stores the measurement into data files so the objects can be studied. Finally, MOA members extract data knowledge from the data files based on association rules, this process is called data mining. Therefore, database system is fundamental facility for astronomy research. Matthew (2002) stated

“Astronomers can use computers programs to find new phenomena, relationships and useful knowledge about the universe and ultimately reduce the gap between data captured and analysis”.

In short, a good database management system can support researchers take advantages of the database to perform data mining.

1.2 Description of MOA data

In its present operation, the data acquired by MOA are saved in flat files. These data include three types.

- **Imaging data.** George (1997) stated “An Image is an optical representation of an object produced by light rays from the object being refracted or reflected by a lens or mirror.” In short, an image is taken by a telescope for stars or galaxies to record what they look like at that moment. These imaging data are useful for a number of research areas, such as detecting asteroids and new earth objects. Hundreds of thousands of individual images (one example is shown in Figure 1) have been accumulated by MOA. Images obtained by MOA telescope are output in Flexible Image Transport System (FITS) format. FITS format files are able to show astronomical images in multiple dimensions. Normally, the FITS format data always comes with “metadata”. Metadata is “data about data” and describes image information. It interprets the basic image data information, such as the number of dimensions in the image and the image size. Because imaging data is very important for the research, these should be saved in the database.

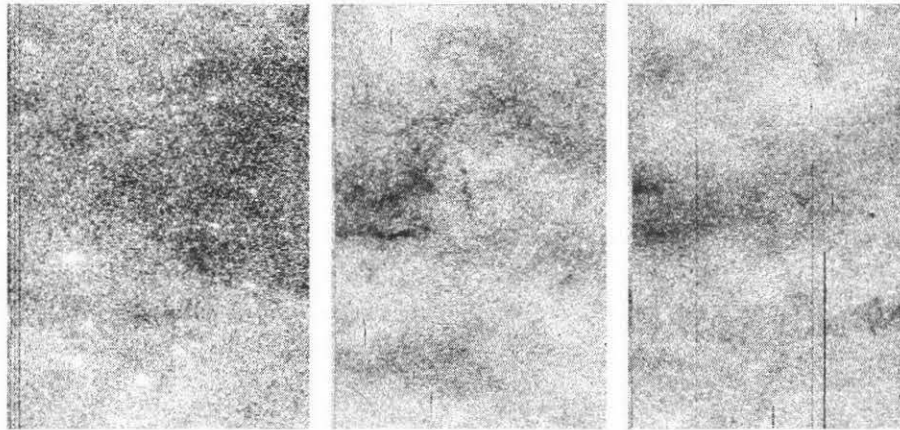


Figure 1: Three images from each CCD of MOACam2

- **Time series photometry data.** These data describe time series intensity measurement data for each star within the field. In the sky, some stars have almost constant luminosity; these kinds of stars are known as constant stars. In contrast, many stars undergo significant variations in brightness over time, and these are called as variable stars. The study of variable stars is of enormous interest to astronomers. American Association of Variable Star Observers (AAVSO) (2006) stated researching on variable stars is important because it

provides information about stellar properties, such as mass, radius, luminosity, temperature, internal and external structure, composition, and evolution. Since MOA members started observation, they have been tracking about 5 millions variable stars. Through studying a long-time behavior of these stars, MOA member can analyze the variable stars behavior. In addition, the variable stars come with a wide range of types and classification. For instance, there are intrinsic variable stars, wherein “variability is caused by physical changes such as pulsation and eruption”(AAVSO) and extrinsic variable star, wherein “variability is caused by eclipse of one star by another or by effects of stellar rotation”(AAVSO). Figure 3 shows an example of just three types of variability.

A light curve is a graph which shows the light intensity of a star over a period. Based on these light curves, astronomers can calculate light curve period, amplitude and magnitude. Then astronomers can use this information to schedule the observation time of the certain stars. Moreover, they can use this information to investigate astronomy science. Therefore, the database needs to record these intensity measurement data.

- **Calibration data.** These data support information on converting object positions from the images to standard astronomical coordinate systems. MOA uses astronomical coordinate system in 2000 to record each object’s location. But the objects’ position is shown as reference position on the captured image. Therefore, astronomers need to translate the objects position from image’s reference value to coordinate system value through photometry and instrument. As a result, the database needs to store both reference position and coordinate position for each observed object in order to support their study.

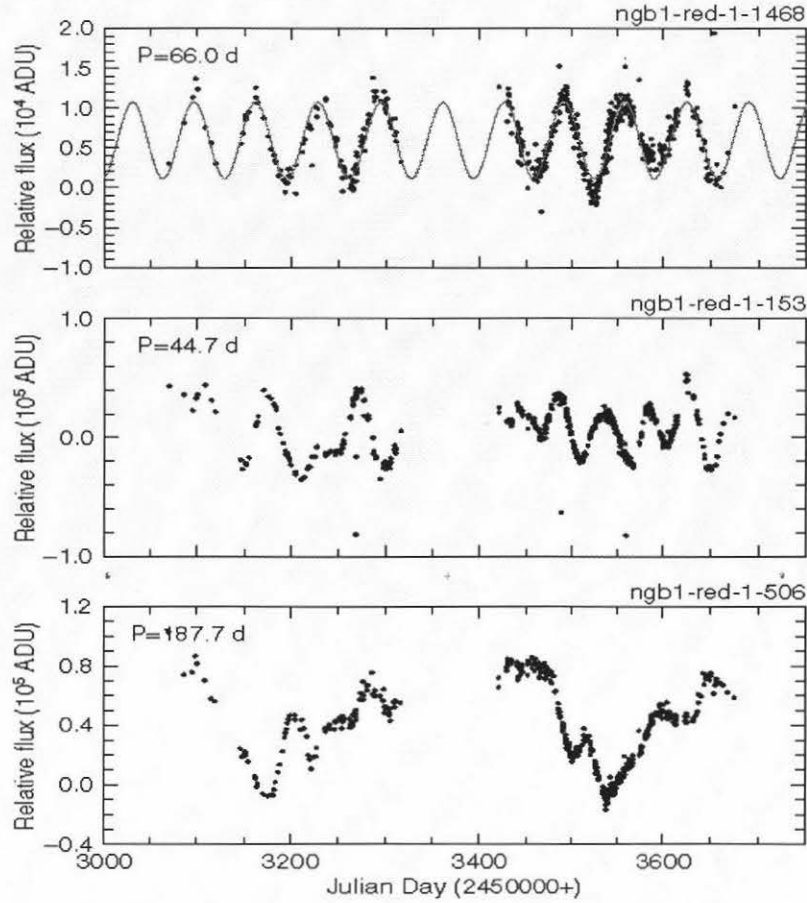


Figure 2: Light curve examples

- (a) Regular pulsating variable star
- (b) Semi-regular variable star
- (c) Irregular variable star

1.3 Current representation of MOA data

Under the current situations, MOA has no real database management system in place although they have made some studies. They use a highly specialized collection of text files and programming scripts to manage the data. However, this is not flexible and is not extensible. Furthermore, some data saved in the flat text files are repeated and it is hard to cross-reference data among different text files. With this state of affairs, it becomes a rather cumbersome operation for any other interested parties to carry out any data mining investigations.

1.4 Overview of this project

Under current situation, MOA needs a new database with database management system to store and effectively manage their data. The database should save astronomy data, support data mining and can be extended easily in the future. In addition, in order to query data easily, MOA needs a good interface to manage the data. The interface can implement users' query on the database and also deliver the query results back to the user. Moreover, the interface can link with research tools such as GNUPLOT. For example, the interface can call the application GNUPLOT with inputting data. So the interface is able to make some of the research operations become pipeline. Then data mining operations will become easier, thereby improving the scientific productivity of the accumulated data.

This project analyzed the modern technology for database and enterprise application. Then designed a suitable application for MOA to manage their tremendous growth data. The prototype supports and benefits astronomical data mining. Due to security consideration, the main aim of enterprise application is to read astronomical data from database and to share them on the Internet. Another stand-alone Java program, it is offline from the Internet, is used for MOA to update and to modify the database (data ingestion). In addition, MOA likes to have a more powerful website to introduce their findings and discuss astronomy events with other on the Internet. It is desirable to have a functional web based interface to the data management system.

2.0 Background on data management

2.1 Introduction of database

Databases play a central position in an information based society. Most of the large information systems have a database as its main part. The usage of database covers from small business system such as payroll system to big enterprise system such as super market checkout system. Kmiec (2002) predicates that over the next decade, people will become increasingly dependent on the correctness and efficiency of the database system.

Roman (1997) describes a database as a collection of records or information that is stored in a computer in a systematic way, so that a computer program can consult it to answer queries. Therefore, the main duty of the database is to store and process records or pieces of knowledge on computer. And the databases are featured in many of application domains ranging from data analysis, modelling, security and data integrity.

2.1.1 Manual Filing system

Manual filing system is an old method for managing data. It offers a way of storing and controlling data. In manual filing system, data is organised in a particular tree-like structure through the use of *directories*. There is no limit on the number of files or directories that a directory can contain. For example, in the past, when there was no computer system, university used a dedicated system to save their essential data. They might save the data in different folders, and then divided the folders into different categories. In particular, all the personal details including lecturers and students were in one folder, and all the business records such as receipts and invoice saved in another folder. When the university staffs wanted to check some information, they went to the right folder and searched them one by one. Alternatively, they might have an index system that helped them to find the location. This is what people used to manage their data.

The manual filing system works well when the number of stored items is small. But with the increase of data size, its performance suffers. In addition, it cannot handle some situations such as cross-referencing among different data sets. Therefore, this system is not suitable for the modern data requirements.

2.1.2 Traditional File-based systems

Before the modern database management system was invented, people always used file-based system to manage data. Thomas and Carolyn (2005) describes a file-based system is a collection of application programs that perform services for the end-users such as the production of report. And each program defines and manages its own data. When the data is requested, the particular application will directly search its own data to find the data. Following the above example, after the university had computer system, it saved its data in the computer files. And there was an application called Personal Information to organize all the personal details. Through this application, the university staffs could add new staffs in the data file; drop staffs from data files and update staff details. However, in the file-based system, if there were multiple applications, each would have its own data file. In other words, applications cannot share data each other. So the university have another application called *business record* to manage the university business records. So the file-based system has big limitations

Limitation of the file-based system

Although the file-based system was a great improvement on the manual filing system, it still has obvious drawbacks, including separation and isolation of the data, duplication of the data, program-data dependence and incompatible file format.

2.1.3 Database approach

From above explanation, we can see the file-based database system has weakness. As a good database system, it should allow data to be shared and controlled between different applications. And the data should be integrated at any time and can be stored separately but be modified consistently. Therefore, a database and the database management system are both important.

The database

A database is collection of data. Thomas and Carolyn (2005) define a database as a shared collection of logically related data, and a description of this data, designed to meet the information needs of an organization. From the definition of the database, we can see all data in database are shared, and can be accessed by different departments and users simultaneously. As a result, the redundant data are optimal to be limited.

Furthermore, when data are saved in the database, they are logically related. When people analyze the organization's activity, they could use *entities*, *attributes* and *relationships* to express the activities. An entity is any objects in the data processing that may be related to any other entities. An attribute is a property that describes the aspect of the object that the system should be recorded. And the relationship is the association between different entities. When the database represents by above three categories, we can say the data is logically related.

The database management system

After storing data in a database, people need software to interact between the application program and the database. This software is called a database management system (DBMS). Thomas and Carolyn (2005) define a DBMS as a software system that enables user to define, create, maintain and control the access to database. Normally, DBMS offers the following facilities:

- Data Definition Language (DDL), such as *create*, *alter* and *drop* command in SQL: this provides users a means to define the database. This language allows users to describe and define the entities, attributes and relationships required for the application, together with any associated integrity and security constraints.
- Data Manipulation Language (DML): it supports to manipulate data in the database. The language provides a set of operations to support users to retrieve, insert, delete and update data in a database
- It provides controlled access to the database, such as offering a security and integrity database system.

Advantage and Disadvantages of DBMS

DBMS has brought both advantages and disadvantages on performance of the database. The advantages include: limits on data redundancy data consistency, supports for sharing the data, and improvements in data integrity and security. And the disadvantages include complexity, cost, reduced performances and higher impact of a failure.

Some types of DBMS

- **Hierarchical:** this supports hierarchical relationships such as parent-child structure between data. Data structures were often forced to conform to the hierarchical model in order to take advantage of the management and programming aspects of the products.
- **Relational:** these are widely used. It uses the Structured Query Language (SQL) to extract and update data and it bases on the rule of normalization to format the database. The relational mode first entered the database in 1970, when Codd (1970) published his seminal work, "A relational Model of data for large shared data banks". The data structure allowed data to be operated in a manner that was predictable and resistant to error. Therefore, the relational model supports a high degree of data independence, accuracy and redundancy.
- **Object oriented DBMS:** This stores data as objects, so it can handle bigger and complex data structure. It provides consistent, data independent, secure, controlled and extensible data management services to support the object-oriented model.
- **Object-Relational:** This system simply puts an object oriented application front end on a RDBMS. When applications interface to this type of database, it will normally interface as though the data is stored as objects.
- **Web:** it is a general term for applications, which provide a web-based interface to database. It supports people to administrate the database through the Internet. The main advantages of Web-DBMS are accessibility, platform independence, easy-to-use, standardized graphical user interface, and transparent network access. But the main disadvantages are reliability, security, cost, and scalability.
- **Hybrids:** it uses many DBMS to handle the complex requirement of the data system.

2.2 Java application

2.2.1 Java technology

Java is a powerful and modern development platform. It has been greatly extended during the last couple of years. Java is showing its power from stand-alone applications to web applications, even mobile system.

2.2.2 Java Enterprise Application

An enterprise application is a broad term that can be broken down into several specifications that together define the whole (Haque and O'Connor, 2002). Typically, the application hosts a server which simultaneously provides services to a large number of users, typically over a computer network.

Java is an efficient way for the enterprise development and it has lots of advantages. In particular, Java is portable and can move between platforms. Furthermore, the Java program, such as Enterprise Java Bean (EJB) can be used by different application servers. For example, an application with EJB can be deployed in different servers when the servers support correct deployment environment, such as JBOSS or SUN application server. The developer Sun Microsystems (2006) said “write once, run everywhere”.

2.2.3 Java Enterprise Application Architecture

With the improvement of Java technologies, enterprise application architectures have experienced a huge evolution. It starts from centralized mainframe structure to multi-tier J2EE architecture. After the application functions can be divided into several tiers, the management of the application becomes easier. For example, Figure 3 is two-tier application architecture and Figure 4 is typical three-tier J2EE application model for Web-based applications. From the Figures we can see, in the two-tier application, it does not separate business logic from presentation logic. It is a client/server architecture, where a request to do some task is sent to the server and the server responds by performing the task. Therefore, this structure causes several disadvantages, such as hard to administer and maintain, and too many

communications on the network. In contrast, J2EE three-tier architecture separates the business logic and presentation logic. The client request is sent to the server and the server in turn sends the request to the database. The database sends back the information/data required to the server, which in turn sends it to the client. Under this structure, each tier dedicates to processing either data or application requests, hence a more manageable system and less contention for resources will occur. Therefore, the three-tier architecture increases component reusability and can handle more complicated applications. In addition, it supports distributed deployment.

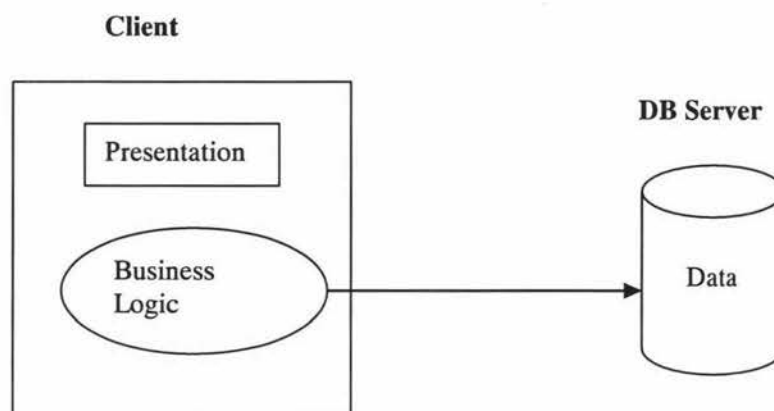


Figure 3: Two-tier Application

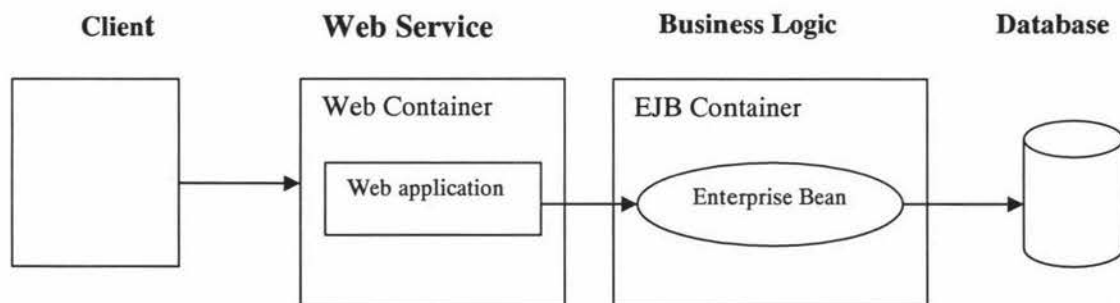


Figure 4: J2EE application Model for Web-based Application

Four different containers build J2EE platform:

- EJB container: it provides a living environment for EJB beans. And EJB are implemented the business processes and entities.
- Web Container: it supports an environment for JSP and Servlet. The JSP and Servlet control the presentation logic of the application.

- Application-Client Container: it provides an environment for executing J2EE application clients.
- Applet Container: the environment for Java applets.

2.2.4 The advantages of J2EE

One J2EE advantage is it makes the development process simpler. It simplifies the control and management of system resources by providing methods to manage transactions and resource pooling (Kifer, Bernstein and Lewis, 2005). As a result, the developers' worries become less. For example, under Java Database Connectivity (JDBC) technology, developers can use automatic data polling to handle the complex connection between application and database. Furthermore, J2EE simplifies the procession of development, configuration and deployment of the application. For instance, it offers consistent environment for developing the application. All in all, on J2EE, developer's job becomes easier.

Another benefit of J2EE development is that it separates the functionality and presentation. That is to say through JSP and Servlet technologies, presentation logic locates in web container layer and business logic resides in EJB layer. Thus, developers can create application functionalities with EJB, and passing the computation results to web container. Then, the web container creates the JSPs to show result to the clients. So the successful separations can isolate the development activities to different parts. As a result, under J2EE, programmers only focus on business logic and designers only concern with presentation logic.

Thirdly, J2EE also supports code reuse. For example, the developers can reuse their EJB. The good reuse ability can enhance understanding and performance of the application.

In addition, the other advantages of J2EE include developer can use open source tool to develop J2EE application and J2EE applications can run in a number of different application servers on many operating systems.

In conclusion, a number of J2EE advantages—such as thread safety, incorporation of other Java libraries, dominant market share, use of design patterns, and awareness of the different technologies involved—can provide developers with valuable functionality.

2.2.5 Design Pattern --MVC in J2EE

Model-view-controller (MVC) is a design pattern used in software engineering. According to Gamma et al (2002) MVC is a way to break applications into three layers, including model, view, and controller. Under MVC architecture, designers have separated data (model) from user interface (view), so that changes of the user interface do not impact the data handling, and that the data can be reorganized without changing the user interface.

At the moment, MVC contains two models, including model 1 and model 2. In the model 1, the application presentation is shown by Html or Jsp files; JavaBean is used to retrieve the data. The JSP page alone is responsible for processing the incoming request and replying back to the client. Also in this model 1, it is page-centric design. All the business and processing logic can either present in the JSP or may be called directly from the JSP page. Moreover, under this model, data access is usually done through Custom tag or java bean call. In contrast, in the MVC model 2, the model removes the page centric property of MVC1 architecture by separating presentation, control logic and the application state. In addition, there is only one controller in the model 2. It receives the request from the application and is responsible for answer each request. Therefore, in the model 1, page and model are tightly coupling; so it is only working well under simple application. The model 2 is represented by JavaBeans, business objects and database, so it is able to handle complex application.

The participants and responsibilities of the MVC architecture are shown as follows.

Model: The model represents enterprise data and the business rules that govern access to and updates of the data. Under J2EE, entity bean can represent the model.

View: The view is the contents of a model. It accesses enterprise data through the model and specifies how that data should be presented. It is the view's responsibility to maintain consistency in its presentation when the model changes. In J2EE, the view

can be represented by Java Server Pages (JSPs). Alternately, the view code can be generated by Servlet.

Controller: The controller translates interactions with the view into actions to be performed by the model. The actions performed by the model include activating business processes or changing the state of the model. Based on the user interactions and the outcome of the model actions, the controller responds by selecting an appropriate view. Normally, under J2EE, the controller can be represented by Servlet.

Figure 5 shows the Model-View-Controller pattern and each layer with functionalities.

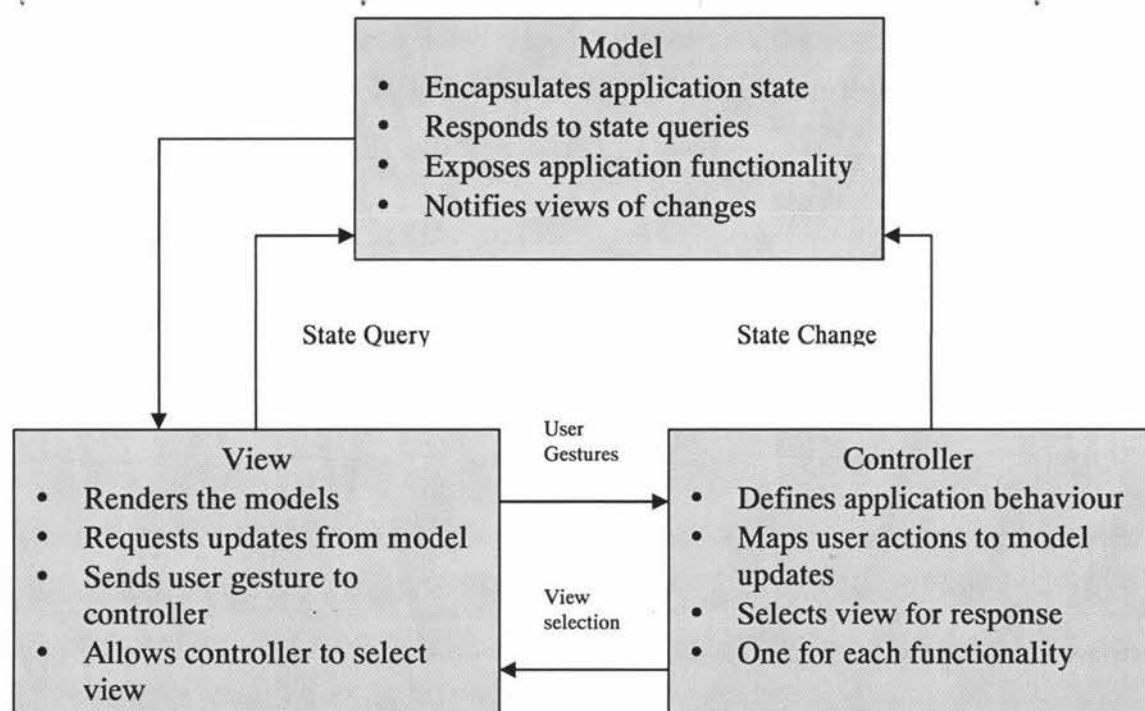


Figure 5: MVC architecture and functionalities

In short, MVC architecture brings both advantages and disadvantages for the applications. Firstly, applications can reuse model components. The separation of **model** and **view** allows multiple views to use the same enterprise model. Consequently, an enterprise application's model components are easier to implement, test, and maintain, since all access to the model goes through these components. Secondly, applications can support new types of clients easily. To support a new type

of client, designer only simply writes a view and the controller logic and wires them into the existing enterprise application. However, MVC will increase design complexity for the application. This pattern introduces some extra classes due to the separation of model, view, and controller.

3.0 Reviews Existing Astronomy Database Technologies and Solutions

3.1 Some Existing Astronomical Projects involving large volumes of data

In astronomical research it is common to study objects in the time domain. For instance, in order to detect microlensing events, astronomers may study one star's activities for several weeks. Therefore, astronomy databases need to record not only historical data, but also words data. In other word, the database needs to be updated frequently. In addition, with the use of modern equipments, the volume of generated data is much larger than before. Therefore, a good DBMS is essential for the research. Normally, astronomers hope that their database not only can support survey operations, storage and analysis the data but also easily communicate with users. Moreover, the database should be easily extended in the future and as fast as possible for retrieving results. In addition, during observations, new information may be generated and need to be saved in the database. Therefore, a good extensibility is essential for the astronomy database.

There are some of the approaches for saving astronomical data.

1. All data are stored in plain text files, and there is a highly specialized scripting application used for organization the data. For example, the MOA group uses this method at the moment. This method supports fast access for a defined set of queries. However, it is not flexible and is not extensible.

2. Another approach that can be used is to design custom astronomical DBMSs, such as DIRA (Benacchio, 1992) or Starbase (John, 1996) to manage data set. Starbase database system can use a simple data file formatting rules and command line data operators to manage astronomical data. Furthermore, it supports for astronomical data and queries, and is freely available. However this does not support large database. And because data is often stored in flat ASCII files, secondary access methods are usually not provided.

3. Relational DBMSs have also been used in the past several years: they are robust systems, widely used in industry, whose data model is close to the structure of astronomical catalogues. And it supports a high degree of data independence, accuracy and redundancy. An example of using RDBMS is shown as follows.

There are two large astronomical surveys in Chile. One is called SuperMacho(SM). The main aim of SM is to detect and follow microlensing events to study the Large Magellanic Cloud. Another group is Essence Supernova. It seeks to detect and follow intermediate to high-redshift supernovae (Chris et al, 2002). During the research, both groups have to explore a big dataset size, which is from GB to TB in real-time.

In order to fit data requirements for the research, both of the groups used RDBMS technology to support time-domain astronomical research. They feel their database is “fast retrieval information, allowing the development of data mining applications and take full advantage of the database” (Chris et al, 2002).

However, it has a limited capability on data modelling. With the astronomical object becoming more complex, it may not model them.

4. Another possible approach is to store astronomical data in Object-Oriented DBMS. These systems feature a powerful data model, which allows data and operations to be modelled.

For instance, Wenger, Kinnar, and Jocqueau (2002) tried to use an OODBMS concept to manage astronomical database SIMBAD. They concluded OODMS can “manage heterogeneous and complex data, and deal with huge collections”

However, OODBMS does not provide an efficient query-processing engine; such facility must be implemented on the top of DBMS (Brunner et al, 2003.).

5. Object-relational approach is a new technique for managing dataset. It supports a user-defined data types and functions to control data. Furthermore, its user-defined index structure can speed up the operation of the query. In addition, an extensible optimizer can lead an efficient way to execute the user queries.

As a pilot research, Baruffolo and Benacchio (1998) build a model based on ORDBMS. After several testing, they concluded ORDBMS could model the astronomical object. And this was possible to extend the database query language with astronomical functionalities and to formulate queries with astronomical predicates.

However, Baruffolo and Benacchio cannot prove the performance of the ORDBMS is better than RDBMS or not. In addition, they found the time for loading data and building index in the database are extremely long.

In conclusion, each storing methods has both advantages and disadvantages. Therefore, the designers should follow the current astronomical data situation and choose the suitable DBMS.

3.2 Possible Database Solution in the market for Astronomy Database

This part simply states some of the possible database solutions on the current market and concludes a suitable DBMS solution for MOA group.

(1). Oracle is a market leader. It has a high-performance. Oracle corporation (2006) said Oracle Database is the most flexible manage enterprise information, develop and deploy business applications. Under the Oracle Database, user can readily respond to changing business requirements, so that improving productivity and reducing downtime. Moreover, it has provided the user-defined data type, which is convenient for the astronomy object. But it is very expensive.

(2). MS SQL server is made from Microsoft and has a similar features to those in Oracle, but it only can be used on the Windows. An example of astronomy research is based on MS SQL is SDSS. Sloan Digital Sky Survey (SDSS) project runs by **Astrophysical Research Consortium** (ARC). The main aim of this project is to create a details multicolour map of the universe. At the moment, they use MS SQL server to manage their data. SDSS stated the total amount of data in their database is more than 800GB and the total number of rows exceeds 3.4 billion. It is an expensive way for managing data. More details about ARC please go the web site: <http://cas.sdss.org/dr4/en/sdss/>

(3). DB2 (Informix). It is a product of IBM. It has the similar feature with Oracle and MS SQL. **NASA Extragalactic Database** is using this technology now. It offers the rich functions on searching astronomy data from the database. But it is expensive as well. More details for NED: <http://nedwww.ipac.caltech.edu/>

(4). Sybase is similar to DB2 and it is traditionally used in many astronomical applications. For example, in Canada, there is a group named The Canadian Astronomy Data Centre (CADC). One important duty of the CADC is to copy astronomical data from the Hubble Space Telescope (HST) to a local server. They use a Sybase database system to save the copied data. More details see:

http://cadwww.dao.nrc.ca/ADASS/adass_proc/adass3/papers/crabbtree/crabbtree.html

(5). MySQL is an open source RDBMS. It is simple, but it does not support customer data type and lacks of supporting on transactions. Also it does not separate transaction log; it does not separate rollback log and does not have file duplication. In addition, it is not extensible and the features are not rich. So it does not suit in complex cases. John (2000) created a web site named "The astronomy Net". The web site was based on PHP with MySQL database. The database saves variety of astronomy data including imaging data and measurement data. The more details please see:

<http://www.astronomy.net/about/history.html>

(6). PostgreSQL is open source and easily extensible. It has good feature sets and supports GiST access methods. An example of using PostgreSQL is Sternberg Astronomical Institute (SAI) web site. The web site saves many kinds of astronomy data sets and offers diversity searching functions.

More details please see: <http://www.sai.msu.su/database.html>

All in all, PostgreSQL is the only extensible free open source DBMS solution. It has powerful set of features well comparable to leading commercial database solutions. Furthermore, it supports object-relational concepts. It allows to custom data type, queries and indexed access methods, optimized for specific tasks. In addition, it supports contribution modules such as pgSphere and pgAstro. PhSphere offers the capability for dealing with geometrical objects in spherical coordination. And the module pgAstro is based on phSphere, it provides astronomy-specific functions and method. Chilingarian et al (2004) stated PostgreSQL appears to be the most versatile DBMS solution for astronomy and astrophysics. With these considerations in mind, PostgreSQL was selected as the DBMS in this study.

4.0 Project Development

4.1 Requirements collections and analysis

4.1.1 MOA database

This section describes the process of MOA collecting data and the data characters, because this information is relevant to the database structure. The observation activities and the obtained data with characters can be shown as follows:

- MOA members divided observation area such as Galaxy Bulge regions and Large Magellanic Cloud LMC, into many small fields. Then each exposure points to one small field. The reason for doing this is all the cameras only can provide a limit resolution on produced images. Therefore, focusing exposure on a small field can make the captured objects become clearer in the image. Figure 6 shows how the MOA member divides Galaxies Bulge into small fields for the current survey. Each numbered rectangle represents the field of view captured in one exposure from MOACam3.

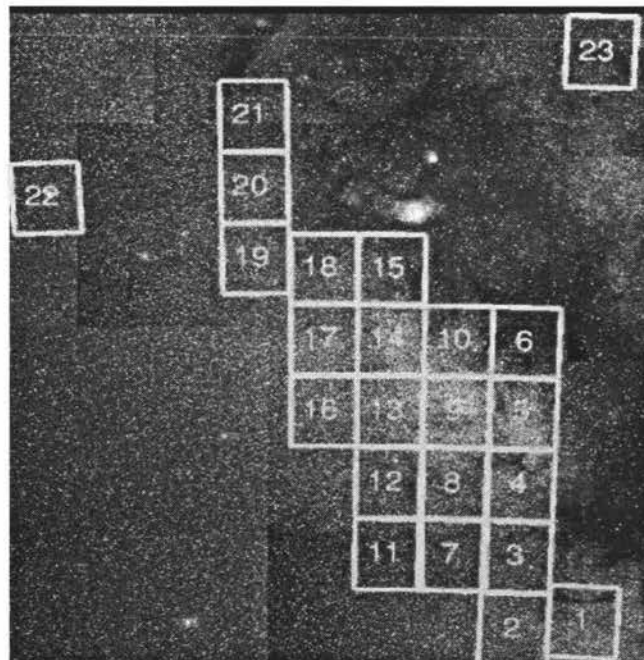


Figure 6: The division of the galaxy in MOACam3

- The calibration data that are used to convert position on the CCD images to positions in standard astronomical coordinate systems.

- In a particular observation, MOA members choose a telescope with camera to take images in the sky. The procedure for taking exposures is based on the camera's CCD construction structure. For example, if the observation uses the camera MOACam1 that has nine CCD chips, in order to make the exposure cover the entire field, four steps builds the observation. The exposure step can be seen in Figure 7. The big square on the Figure is the observation field and the exposure starts from the position P1. After taking photos, the camera moves to the area P2, and then it changes to P3, P4.

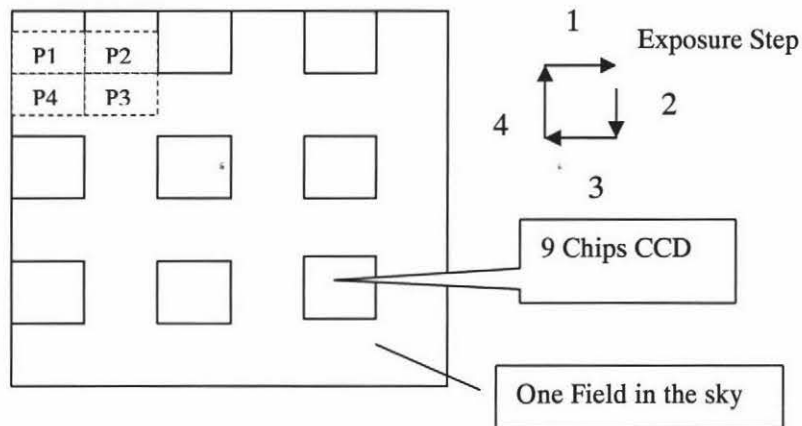


Figure 7: The telescope exposure step in MOACam1 in a field.

- The typical information is generated during exposure includes observation location (area of the sky), observation duration time. Actually, they are part of imaging metadata. The imaging metadata includes Run number (an identification for a particular observation), Field number (the location of the sky), CCD (which CCD on the camera), Colour (the filter of the camera), Camera (which camera is used), Exposure step in one field and the exposure time.
- A RUN shows a specific sequence of target fields for exposure. Typically, a run lasts for a few hours.
- The observation produces a large amount of images.
- Occasionally, the telescope takes images out of the intended field. Under this situation, the captured images should be adjusted through the value Position shift X and Position shift Y.
- The information extracted from the images includes each star's position - Position X, Position Y (the star's reference position on the image), RA, DEC (the star's position in calibration system), Time intensity value (composite by obtained time,

run number and the Flux value), Error (the noise of the intensity value), INDEX (one star's identifier in one image) and the Number 1, Number 2, Number 3, Number 4, Number 5 for recording some other star's properties.

- MOA has been tracking about 5 millions stars for more than ten years and each star has a big number of time intensity values.
- Each star can be identified by composite value, including Field (area of sky), CCD (camera chip), Colour (camera filter), Camera (camera used) and Index (index number on the imaging data).
- Each star' has its own properties such as its light curve period, amplitude and classification. This information is important for MOA to do data mining.

The observing process is shown in Figure 8.

1. MOA staffs decide which telescope and camera will be used in the observation.
2. The camera takes exposure in the sky and produces images with image metadata.
3. The information on the images can be digitized into measurement data.
4. The measurement data and images are saved
5. MOA members do data mining based on these numbers and images. During data mining process, MOA can find new data knowledge or use existing data knowledge in the research.

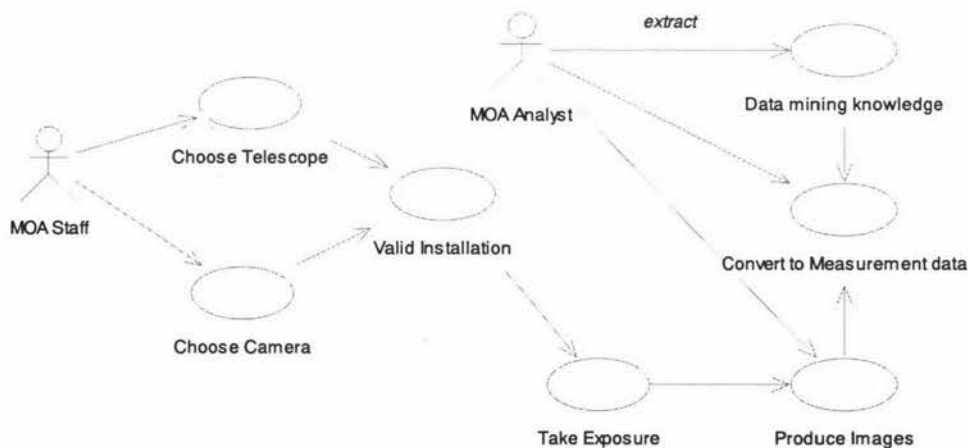


Figure 8: The diagram for observation activities

Typical Query Transactions

The purpose of the database is to save data and support data mining. Some possible queries that could be used in data mining operations are listed below.

- Get a particular star's intensity data within a particular period.
- Find one star's neighbour stars.
- Get star according to star's RA and DEC value.
- Retrieve star according to their classification, light curve period properties.
- Know any exposure's details such as start, end time and counting the duration of the exposure.
- Find an imaging file based on special conditions through image metadata.
- Show any observation field with location value (RA and DEC).
- Count how many stars in one observation field.
- The details of the telescope and CCD cameras.

4.1.2 The web application

The main objective of the web application is to access to the database and to share the large volume of data with other through the Internet. Moreover, this application is not only a database management system (DBMS), but also can support data mining such as using data knowledge to exam data. As a result, users are able to study astronomical events based on the application. Therefore, the application should have a good performance on interacting with the database. In addition, the application should introduce MOA group to other on the Internet, as a powerful web application it should have ability to handling hundreds even thousands of users' concurrent request.

In order to include MOA requirements, after an iterative process, the web applications features should include:

1. A front end to manage the database.
2. General introduction of MOA
3. Introduction of MOA observing technique: gravitational microlensing
4. Introduction of MOA publications
5. Introduction of MOA's observation equipments: telescopes and cameras
6. Allowing client register as member and having privileges.

7. Accessing to MOA astronomical database and then querying wanted data
8. Support for interactive drawing of light curves for individual stars. And the pictures are generated by GUNPLOT.
9. Providing members to download astronomical data in a text file from the database.
10. A discussion board for users to discuss and share their ideas.

4.2 Application Architectural design

From above introduction, we can see the MOA members will manage the database and the data will be shared on the Internet

Characteristics of MOA database include:

- The database should afford a big volume of datasets.
- The database is able to load new data.
- Once the scientific data are recorded, they won't be modified any more.
- There are no complex relationships among datasets at the moment. But in the future it may have.
- The database should process data quickly because MOA members need to analyze current data and bring the result to configure their observation.
- The data can be saved in a centralized database.

Characters of the application are:

- It can manage the database.
- It can be treated as a website so it has website features. Some of the requested features can be shown by static web pages such as the web page of introduction of MOA project. And others must be shown by dynamic web pages such as querying data from database.
- Only MOA members can update the database through the application, but other is able to read astronomical data.
- The web site may face a wide range of visitors.

Based on these characters, the database would be created under PostgreSQL and it would be modelled by relational model, because the relational model supports a fast access to data and maximally reduce redundant data. And the enterprise application

design followed MVC pattern under J2EE architecture. The reasons includes Java Bean supports the high abilities on processing data and modelling enterprise activities, JSP is able to show both static and dynamic contents easily on the web pages and JDBC easily control the data transformation between the database and the application. The application would include two independent Java programs. When MOA members update the database, such as data ingestion and updating, the first stand-alone program would be used. This program will not share on the Internet. Another enterprise application would work through the Internet; it is a front end for user to query data from the database. But this application forbids users to modify the database. So it ensures hacker cannot attack the database through the Internet.

4.3 Database Design

This part presents an overview of the main approach in the database design. In order to clarify the meaning of the data and to facilitate communication between activities, relational data model was used to describe the data, relationships and constraints between data. The database design is made up by three main phases, including conceptual, logical and physical design.

4.3.1 Conceptual database design

According to the description of the MOA observation activities, the conceptual data model for the database was concluded. Thomas and Carolyn (2005) state the conceptual model is conceptual representations of the database. It points out all entities with relationships and attributes in the activity and the kind of keys among entities. The procession is simply divided by three steps.

Step1: Identify entity

This step defined the main objects which the observation activities are related to and then clarified the functionalities and role of these objects. Based on observation process, the entity types can be seen in the Table 2.

Table 2: MOA entity types

Entity name	Description	Occurrence
Telescope	General term describes all telescopes that MOA has	Each observation only use one telescope
Camera	General term describes all cameras that observation can use	Each camera used to combine with certain telescope.
Exposure	It describes MOA's exposure activities. It can show each exposure's details such as exposure's start and end time, shooting area. It can produce the metadata for the image data.	Each exposure has certain identifier. And it has different shooting rules dependent on the camera CCD detector.
Image	It stores the obtained imaging files with imaging metadata. Each image shows particular stars' properties, such as flux and error value in a certain time	Each exposure produces many images
Star	General term describes each observed star's properties such as star's type, location, time intensity value, light curve period and classification.	Each star is an observation object in the sky. The star's time intensity values are shown by the image measurement data.
Pointing	It describe the situation when telescope is not exactly points to granted location in the sky during exposure	This phenomena only happens occasionally
Calibration	It describes the exposure's location	Each field has own calibration data.

Step2: Describe the relationship among entities.

This step is used to identify the relationships among entities.

Table 3: Relationships among entities

Entity name	Multiplicity	Relationship	Multiplicity	Entity name
Telescope	1	Install	1	Camera
Camera	1..1	Take	1..*	Exposure
Exposure	1..1	Produce	1..*	Image
Image	1..1	Show	1..*	Star time intensity value
Exposure	1..1	Meet	0..1	Pointing
Exposure	1..*	Look up	1..*	Calibration data
Star	1..1	Have	0..*	Time intensity value

However, through analyse, it was decided that it was not necessary to save imaging data in the database. The reasons are shown as follows. Firstly, the high quality

imaging data has an extremely big size. Therefore, these large objects will occupy a large memory in the database. Secondly, the information on the imaging can be transferred into measurement data. Because the most of data mining methods are based on these measurements data, saving these relevant data is able to support for research. Lastly, image operations are time consuming in the database. It spends a long time for saving image in the database and reading them from database. Therefore, the designer decided to save the image data in the hard disk as image file and save the imaging metadata in the database tables. The image metadata contains the interpretation of image, such as image name, captured telescope and image position in the sky. Such information plays an important role when searching for and browsing images. This information is derived from the exposure process and saved in the entity **Exposure**. In the future, researcher can create an application to search imaging data based on imaging metadata from database table and then an new application can be used to retrieve the matched imaging FITS files from the hard disk. In addition, a new attribute called *time intensity value* was created under the entity **Star** to save the time intensity value for the each star. The time intensity values are the measurement data of stars and are derived from the imaging data. Obviously, this attribute has multi-values.

Step3: Determine attribute and keys

This step is to decide the attributes and primary key for each entity.

Telescope: TeleName (PK), Length

Camera: Cameraid (PK), step, Naxis1, Naxis2, NumberCCD

Exposure: Run, Colour, Shooting position (Composite by: Field, CCD, Camera, exposure step) (Composite PK), JDstart, JDend

Pointing: Run (PK), Pos shift X, Pos shift Y, a, b, c, d,

Calibration: Sky Location (Composite by: Field, CCD, Camera) (PK), RA, DEC, T1, T2, T3, T4, T5, T6, T7, T8

Star: Field, Colour, CCD, Exposure step, Camera, Index (Composite PK), Star Location (composite by: reference position on the image, position on astronomical coordinate system (RA and DEC), position shift x, position shift y), Period,

Classification, Time Intensity value (composite by Time, Run, Flux, Error, N1, N2, N3, N4, N5)

In these entities, composite primary key was used in the table **Exposure**, **Calibration** and **Star**. A primary key used to identify a particular record in the table. However, sometimes it requires more than one attribute to uniquely identify a record. Mike (2004) stated a primary key that made up of more than one attribute is a composite primary key.

The conceptual model for MOA database is shown by Figure 9. (The entity **Star Property** on the Figure 9 is the entity **Star**). From it we can see the relevant entities, the entities relationships and the attributes should be involved in the database.

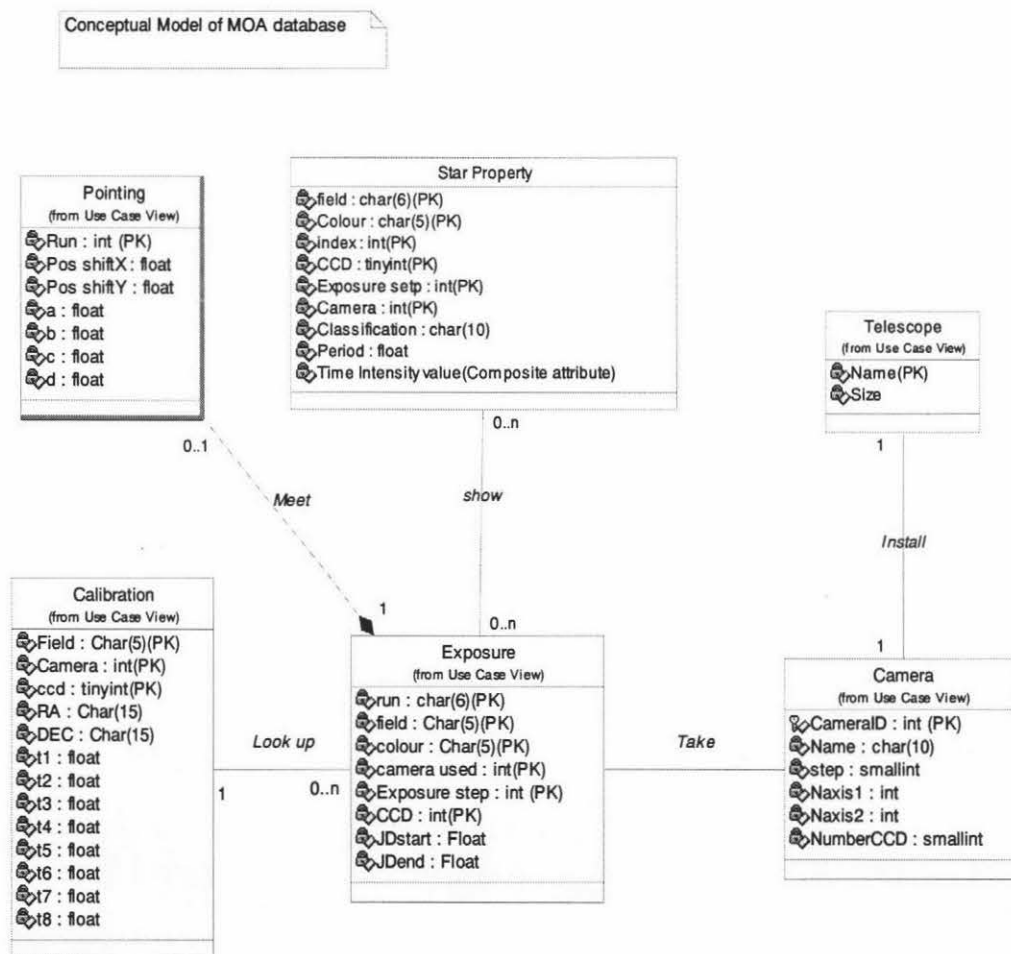


Figure 9: The conceptual model for MOA database

4.3.2 Logical database design

The purpose of this stage is to translate the conceptual data model into a logical data model. This step is independent from particular DBMS and other physical considerations. Some methods were used, such as normalization of relations, to validate the data model. As a result, the model is structurally correct and is able to support for MOA required transactions.

Step 1: Derive relations for the logical data model

This steps derived the relations from the conceptual model and shown the entities, relationships and attributes. In the relational data model, the relationships among entities are represented by primary key (PK) and foreign key (FK) mechanism. In order to decide where to place the PK and FK, the model needs to identify the relationship type (parent and child) among the entities. Then following the relationship types, the model can post the PK and FK in the proper places.

For instance:

One-to-one binary relationship (Mandatory participation on one side of a 1:1):

The relationship can be classified by parent and child entities. In order to derive this relationship, the designer can put a copy of the primary key from parent entity into the child entity. A simple example can be seen in the Figure 10.

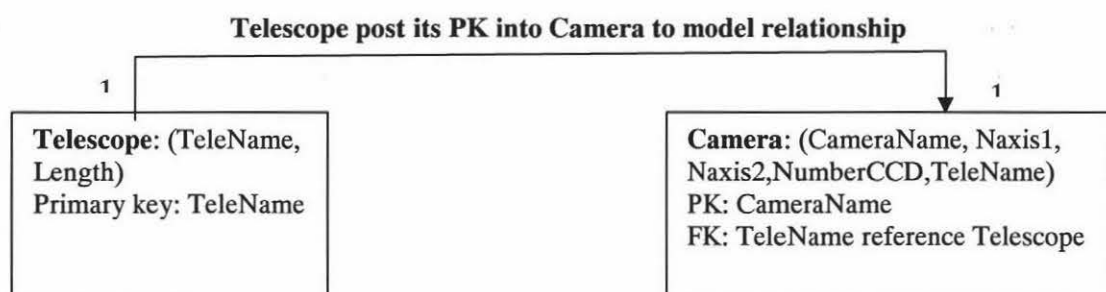


Figure 10: Derive the relationship between entity **Telescope** and **Camera**.

Each telescope only picks one camera each time.

One-to-many (1:*) binary relationship:

For each 1:* binary relationship, the entity on the 'one side' of the relationship is designated as the parent entity and the entity on the 'many side' is designated as the

child entity. In order to derive this relationship, the designer can post a copy of the primary key attribute of the parent entity into the relation representing the child entity to act as a foreign key. Figure 11 shows an example of this relation.

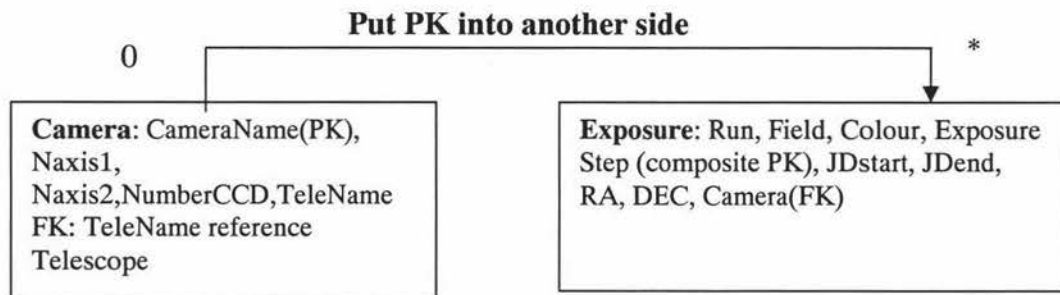


Figure 11: Derive the relationship between **Camera** and **Exposure**

Each Camera can produce from 0 to many exposures and each exposure only can use one Camera.

Multi-valued attributes

As discussed early, the attribute *Star Time Intensity* is a multi-valued attribute in the entity **Star**. For this kind of relationship, we can create a new relation called **Star Intensity** in this case to represent the multi-valued attribute. Then primary key was posted from the entity **Star** to the new entity **Star Intensity**. A simple example can be seen in the Figure 12.

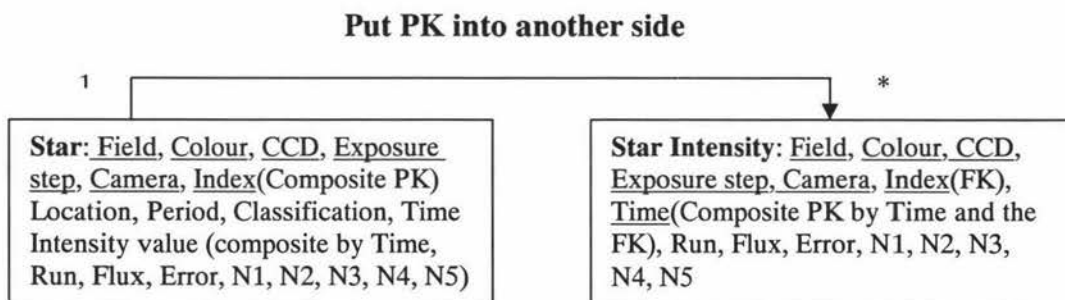


Figure 12: derive the relationship between **Star** and **Star Intensity**

Step 2: Validate the relations using Normalization.

So far, a simple model to describe the observation activities had been concluded. However, some of the data representation or the relations may not show the activities accurately. Or the data model has data redundancy or update anomalies. Therefore, a database design technique known as “normalization” was used to examine the data model. Normalization uses a series of methods to identify the optimal grouping for the

attributes. It produces a set of suitable relations, which supports the data requirements of the enterprise. The formal definition of the normalization is “a technique for analyzing relations based on their primary key (or candidate keys) and functional dependencies” (Thomas & Carolyn, 2005). The process of the normalization can be divided into several steps.

Unnormalized Form (UNF): a table that contains one or more repeat groups.

First Normal Form (1NF): a relation in which the intersection of each row and column contains one and only one value.

Second Normal Form (2NF): a relation that is in First Normal Form and every non-primary-key attribute is fully functionally dependent on the primary key.

Third Normal Form (3NF): a relation that is in First and Second Normal Form and in which no non-primary-key attribute is transitively dependent on the primary key.

Based on Normalization technique, the function dependencies were clarified among tables. Function dependency is an important concept for the normalization. It describes the relationship between attributes. Once we know the attributes relationships, we can separate them from the table and reduce the data redundancy. Figure 13 shows the function dependencies for the table Star and Calibration. And Figure 14 shows the transformation process of the table **Star** after normalization.

Table Star:

Field	CCD	Index	Camera used	Exposure Step	Period	Colour	Classification	Reference_pos X	Reference_pos Y	Shift X	Shift Y	RA	DEC
-------	-----	-------	----------------	------------------	--------	--------	----------------	--------------------	--------------------	------------	------------	----	-----



Table Calibration:

Field	CCD	Exposure Step	Camera Used	RA	DEC	Other values
-------	-----	------------------	----------------	----	-----	-----------------



Figure 13: Function dependencies of the table **Star** and **Calibration**

Table Star before normalization:

Star:

Field	Colour	Exposure Step	Camera used	CCD	Index	Classification	Period	Position X	Position Y	RA	DEC	Shift X	Shift Y
-------	--------	---------------	-------------	-----	-------	----------------	--------	------------	------------	----	-----	---------	---------

Table Star after normalization:

Star:

Field	Exposure Step	Camera used	CCD	Index	Classification	Pos X	Pos Y	RA	DEC	Shift X	Shift Y
-------	---------------	-------------	-----	-------	----------------	-------	-------	----	-----	---------	---------

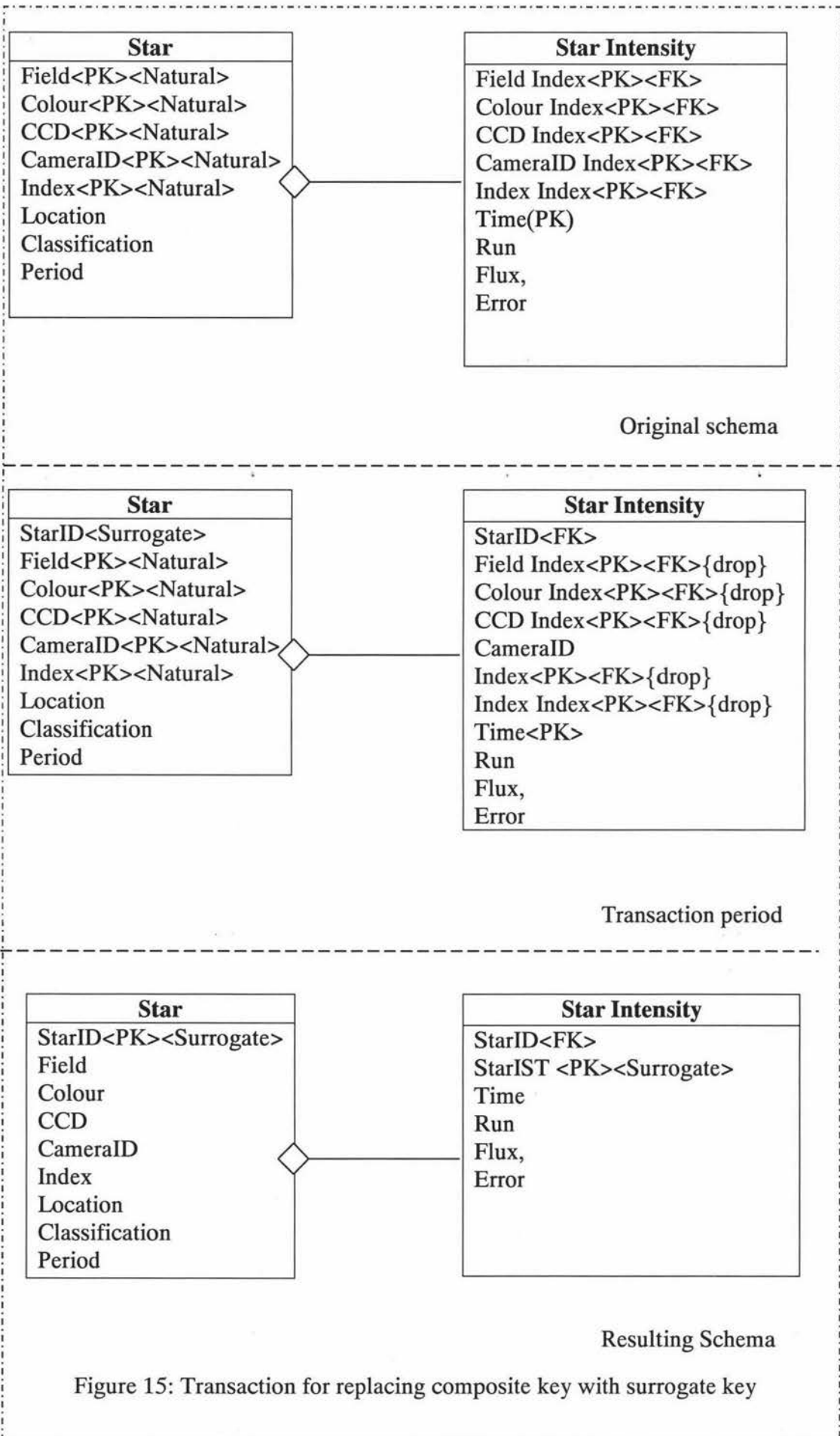
Star periodicity

Field	Colour	Exposure Step	Camera Used	CCD	Index	Period
-------	--------	---------------	-------------	-----	-------	--------

Figure 14: The table Star before and after Normalization

Step 3: Replace Composite Primary Key with single primary Key. In the table **Star**, **Star Intensity**, **Calibration** and **Star period**, there are composite key as primary key for each table. But composite primary key has drawbacks. Firstly it slows the database performance considerably, as join operation needs to check all contents of all columns designated in such a composite key. Secondly, the sequence of the fields concerned must be identical in all referenced tables.

In order to avoid using composite primary key, a surrogate key was used to instead of them. The transformation operation applied in the table **Star**, **Star Intensity** and **Exposure**. Surrogate key is a numeric column and it automatically increases when new data is inserted. Therefore, it is completely unrelated with the data. The Figure 15 shows the relationships and attributes in the table **Star** and **Star Intensity** before and after applying for surrogate key.



However, after introduced surrogate key in the table, the SQL command for inserting and updating record in the table becomes harder than before. For example, if people like to insert one star's time intensity value into the table **Star Intensity**, first of all they should retrieve the foreign key *StarID* value from the table **Star** for this star. Then based on the *StarID* value, people can insert the new record.

An example is shown below for inserting a record into database before and after the table applies for surrogate key.

The query for inserting star's time intensity value into the table **Star Intensity** is:
For example, the star identifier is "Field is 1,ccd is 3, index on the image is 100, filter is red and cameraID is 5"
Before applied for surrogate key, the SQL is:
Insert into StarIntensity (field, CCD, Colour, index, CameraID, flux) values ('field1', 3,'red', 100, 5, 9879);

After applying for surrogate key, the query becomes:
Firstly, retrieve *StarID* value from the table **Star**:
Select StarID from star where field='field1' and ccd=3 and cameraid=5 and filter='red' and index=100;
Let's assume from above query, we know the *StarID* is 5 for this star in the table **Star**.
Secondly, insert data into the table **Star Intensity**. The query is:
Insert into StarIntensity (StarID, flux) values (5, 9879);

From above example we can see, after the table applying for surrogate key, the data ingestion will spend more operations than before. However, the query performance of join the tables will be faster than before due to simplified join key. Because the main aim of this database is to support data mining, a good query performance is the most important point for the database. Therefore surrogate keys were applied in these tables.

In addition, the column *Camera used* and *Exposure step* are together to constraint a particular exposure in the observation. In order to save space and enhance the

performance, these two columns were merged to one column called CameraID. CameraID included all possible combination among *telescope*, *Exposure Step* and *Camera*. Table 4 shows how the CameraID value mapping to Exposure step, Camera and Telescope.

Table 4: The new relationships among attributes

CameraID value	Camera	Exposure step	Comprised telescope
1	MOACam 1	Step 1	0.6 m
2	MOACam 1	Step 2	0.6 m
3	MOACam 1	Step 3	0.6 m
4	MOACam 1	Step 4	0.6 m
5	MOACam 2	Step 1	0.6 m
6	MOACam 3	Step 1	1.8 m

Step 4: Check integrity constraints

This step checked integrity constraints from required data, attribute domain constraints, multiplicity, entity integrity, referential integrity and general constraints to guarantee the database saves correct data.

Step 5: Consider the Introduction of Controlled Redundancy

Sometimes, introducing redundancy in a controlled manner by relaxing the normalization rules can improve the performance of the system. A normalized database prevents functional dependencies in the data so that updating the database is easy and efficient. However, querying the database might require many joins of tables to combine information. The more join operations include tables, the longer time will be spent on procession the query. Therefore, a normalized database might not always be the best choice. A database with the appropriate amount of denormalization reduces the number of tables that have to join together, without adding too much complication to the updating process. This is frequently a good compromise.

In this database model, the table **star** and **star periodicity** can be combined together. So the table calibration was denormalized. Also the date in the table **telescope** and

camera can be saved in one table. The new table was called camera. After converting, the query for updating and joining the table become easier.

This step completes the logical database design. Figure 16 shows the logical model of the database and table details can be seen as follows.

(Postscript: The attributes Field, Cameraid, CCD and Index are the star identifier.)

Table name: *Star*

Table attributes: StarID(PK), Field, Cameraid, CCD, Index Colour, Period, Classification, RA, DEC, Pos, Pos Shift

Description: The table saves each star's properties.

Attributes Period means the light curve period of this star.

Classification stands the type of this star.

RA, DEC, Pos and Pos Shift describe the stars' location in different scale system.

In addition, more stars' attributes can be appended here in the future.

Relevant data file: ".par.dat" file

Table name: *Starintensity*

Table attributes: starIst(PK), starID(FK),time, Indx,run,Flux, Error, N1,N2,N3,N4,N5,ExposureID(FK)

Description: The table stores all stars time intensity measurement data. As a surrogate key, staristID is the primary key for the table.

StarID is a foreign key and is referenced from the table **Star** and the foreign key ExposureID is referenced from the table **Exposure**

Relevant data file: Varplot file-".diphot.xml"

Table name: *Exposure*

Table attributes: ExposureID(PK), CameraID(FK),CCD(FK), Field(FK), Colour, Run, JDstart, JDend

Description: The table has each exposure details. It saves the metadata of the imaging data. JDstart and JDend record when a particular exposure starts and ends. And more imaging metadata can be appended here for description

the imaging data details, such as imaging position in the sky and the size of the image.

CamerID, CCD, Field are foreign keys from the table **Calibration**

Relevant data file: “.info” file

Table name: *Pointing*

Table attributes: ExposureID(PK)(FK), Pos Shift X, Pos Shift Y, a, b, c, d

Description: The table stores the particular pointing event data.

ExposureID is a foreign key referenced from the table **Exposure**.

Attributes Pos Shift X, Pos Shift Y, a, b, c and d are the values supporting for retrieving the correct star position

Relevant data file: “.match” file

Table name: *Camera*

Table attributes: CameraID(PK), Name, Step, Naxis1, Naxis2, NumberCCD, Telescope

Description: The table records camera’s information with bound telescope

Table name: *Calibration*

Table attributes: Field(PK), CCD(PK), CameraID(PK), RA, DEC, t1, t2, t3, t4, t5, t6, t7, t8

Description: The table is a “look up” table; it saves the calibration data for each field.

It uses composite key including Field, CCD, CameraID as primary key.

Relevant data file: “.par” file

Logical Diagram for MOA database

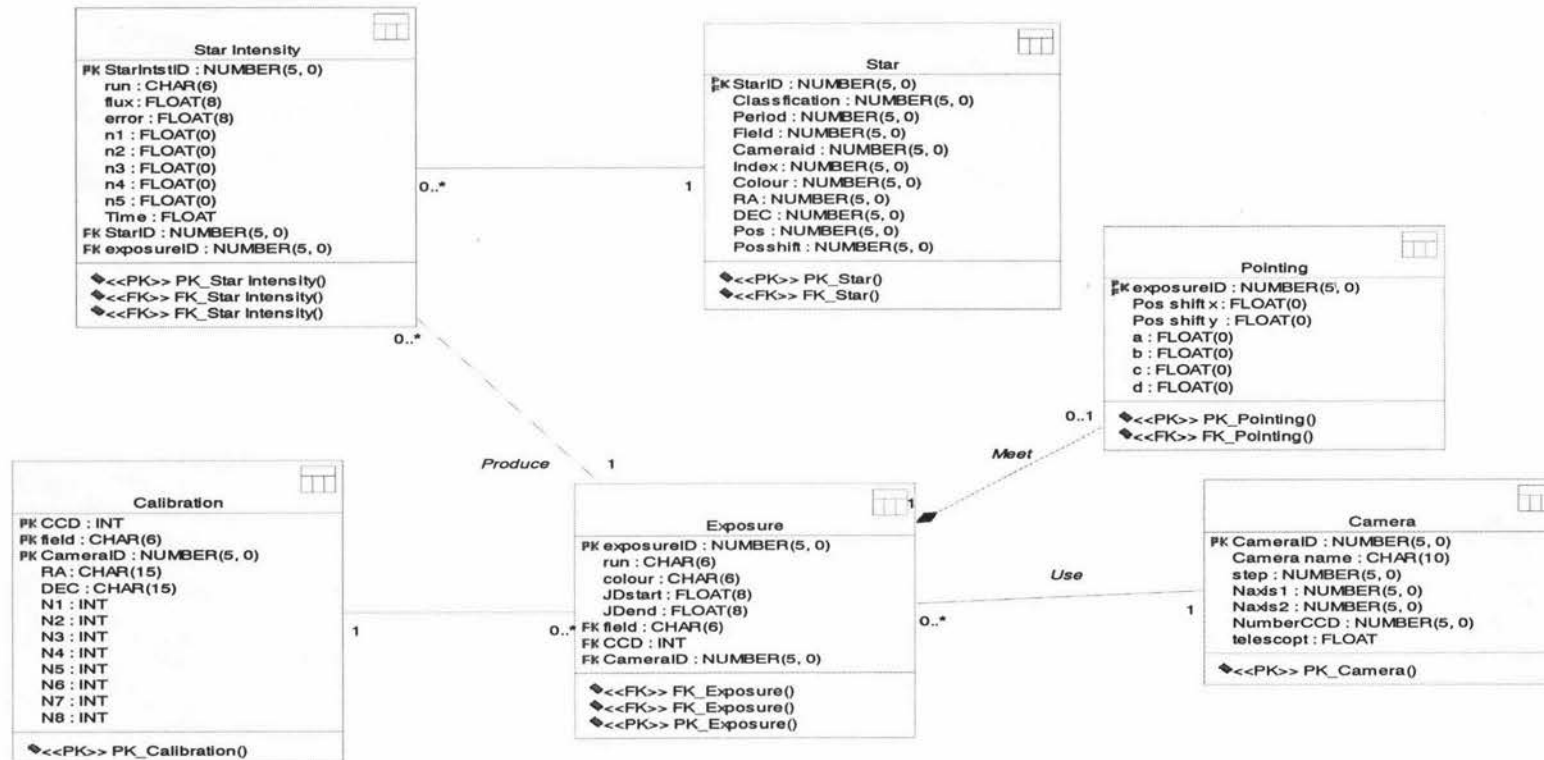


Figure 16: Logical model for MOA database

4.3.3 Physical database design

So far we have concluded the logical database model for the database. Now let's go to the next step: physical database design. Physical database design is

“The process of producing a description of the implementation of the database on secondary storage; it describes the base relations, file organization, indexes, and any associated integrity constraints” (Thomas and Carolyn, 2005).

The physical database design was divided into four steps.

Step 1: Translate logical data model for target DBMS

This step translated the logical database model into database schema. So the model can be presented by SQL command under PostgreSQL. The PostgreSQL commands for creating database can be seen in APPENDIX A.

There are some points that were noticed during the design.

1. Use most suitable type for each attribute. In the model, there are many types to define each attribute. For example, the attribute **Time** in the table can be defined as Character or Float. The attribute **Time** will be used frequently for calculation and comparison in the future. Because numerical type can provide noticeably better performance than character type on these operations, the **Time** took float as its type.
2. Never waste database space. Because it will be a big volume of data sets in the database, the appropriate size is essential for the attributes. For example, if the content of an attribute is up to 10 characters in the table, this attribute size was defined as 10. In contrast, if the column is defined as size 20, it means the database waste space on it. With the data sets becoming bigger and bigger, this would involve some wastage of database space.
3. Design the integrity constraints for the attributes. Integrity constraints can ensure that the data is accuracy and consistency in the database. In this model, they were created according to MOA data meaning and entity relationships. An example of defining data constraints under PostgreSQL can be shown as follows.

Command for referential constraint:

```
create table disdetails (detailsid serial primary key, id int not null, author char(30)
NOT NULL default "", time timestamp, content text, foreign key (id) references
discussion (id) on update cascade on delete cascade);
```

Command for unique constraint:

```
create table star (starid serial primary key,field char(6),ccd int,index int,colour
char(5) default 'red',cameraid int,period float default 0, classification char (12),ra
char(15),dec char(15), pos point, posshift point,
unique(field,ccd,index,colour,cameraid));
```

Step 2: Design indexes

One of the objectives of the database is to access its data efficiently and effectively. If the data saving follows the sequence of its future use, the future operation will save much time. In addition, index is a way to speed up user queries. But it also spends disk space. Therefore designers only set up indexes when it is really necessary. This project predicated some popular queries which might be used by the researchers in the future. Then index was created according to these activities. PostgreSQL supports several index types such as B-tree, hash and R-tree. B-tree index fits the most common situations. It can handle equality and range queries on data that can be sorted into some ordering. (PostgreSQL global development group, 2007). And the Hash index is able control the simple equality comparison operation.

The location is an important character for stars. When MOA members do the research, they often select one or a group of stars according to their location or a location range. Therefore, in order to improve the database performance, a B-tree index was created for the star location. The PostgreSQL command for creating index is shown below.

```
Create index locationid on star using btree (RA, DEC);
```

When MOA members do the data mining, they need to research light curves. Therefore, it is necessary to set the star's identifier and observed time as B-tree index in the table to enhance the database performance when they query the data.

The PostgreSQL command for creating the index:

```
Create index starid on star using btree (field, CCD, colour, cameraid, index);  
Create index timeid on starintensity using btree (time);
```

Querying light curves is an important operation for astronomy research. So one hash index called `star_classification` which is relevant to star's classification was created. And one B-tree index `star_period` which is related to light curve period in order to accelerate the database performance.

The SQL command:

```
Create index star_classification on star using hash (classification);
```

The SQL command:

```
Create index star_period on star using btree (period);
```

Step 3: Design Views

A VIEW is a popular technique used in the data management. View is a dynamic result of one or more relational operations on the base relations to produce another relation. View is virtual relations but not real exist in the database (Thomas & Carolyn, 2005). Normally, view can join many tables together and show their data relations. The advantages of view include data independence, currency, improved security, and reduced complexity. This project created a view called `view_star` for joining each star with its time series intensity data together, because these information are always used together but among different tables. When the new data is inserted, the view should be updated.

```
create view view_star as select  
s.field, s.ccd, s.colour, s.cameraid, s.period, s.classification, s.index, i.flux, i.error, i.run,  
i.time from star as s, starintensity as I where s.starid=i.starid;
```

Based on this view, the query for selecting data from these two tables become:

```
Select * from view_star where +condition;
```

Step 4: Backup and Recovery design

Backup database is an important phase. Thomas and Carolyn (2005) stated backup is the process of periodically taking a copy of the database and log file on to offline storage media. The PostgreSQL command **pg_dump** was used to backup the database, because it can make a copy of the database. Furthermore, **pg_dump** can write data to

the standard output, so it can perform large database backup. When the database meets equipment failure or disaster, it can recover or retrieve the database based on the backup files.

PostgreSQL command for backup database using compressed dumps (under shell with gzip compression program):

```
pg_dump dbname | gzip > outfile
```

Command for reload the database through gzip compressed backup files:

```
Gnuzip -c backupfilename dbname < infile
```

In addition, when the database makes major changes to the contents of the table, such as inserting a big volume data and changing table structure, it is a good idea to run command **ANALYZE**. This collects statistics information about the contents of tables. The query planner can use these statistics information to decide the efficient execution plans for query. So it can enhance the database performance.

4.4 Data Ingestion application design

After the database structure has been decided, it needs to change the source data format and then loads them into the database. As mentioned before, due to security consideration, a Java program has been written to perform data ingestion. The interface of the program can be seen in the Figure 17. Seven classes have been developed and each of them provides different functionalities.

```
=====
MOA database loading system
-----
1. Load the star location (.pos) file into the database
2. Load light curve data (.dat) into the database
3. Load calibration data file into the database
4. Load exposure (.info) file into the database
5. Load pointing data into the database
6. Load time intensity value (.diphot.xml) file into the database
   Type 'end' to EXIT the system
Please choose your operation and press ENTER
=====
Enter your choice here:|
```

Figure 17: The interface of the ingestion application

A full source code implementation can be found in the enclosed CD. This section provides dissection of the classes.

Java Class: Calibration

Functionalities: It reads data from the source file extension with ".par" and then it loads the data into the table **Calibration** through bulk loading.

SQL statement:

```
SQLQuery="insert into calibration  
(field,cameraid,ccd,cali,n1,n2,n3,n4,n5,n6,n7,n8) values ('granted value');
```

Java Class: Exposure

Functionalities: It reads data from ".info" file and then it loads the data record into the table **Exposure**.

SQL statement:

```
SQLQuery ="insert into exposure (run,field,colour,jdstart,jdend,cameraid,CCD)  
values (granted value);
```

Java Class: Starlocation

Functionalities: It reads data from *star location* source files and then it loads the data into the table **Star**. The data ingestion application uses this kind of source file to initialize each star's record in the table **Star**.

SQL statement for inserting data:

```
SQLQuery=insert into star (field,CCD,index,cameraid,pos,posshift,ra,dec) values  
(granted value);
```

Java Class: Period

Functionalities: It reads data from *star periodicity* source files and updates the relevant star record in the table **Star**. After the class Starlocation has initialized each star's record in the table **Star**, this class is used to update designated star's period property.

SQL statement for inserting data is composed by two steps:

1. Retrieve the relevant *starid* value for current star from the table **star**. (Let's assume the variable index, field, CCD, and cameraid have held the appropriate value for a star.)


```
SQLquery="select starid from star where field='ngb'+field+" and  
index="+index+" and ccd="+ccd+" and cameraid="+cameraid;
```

2. Update this star's information (Let's deem the variable starid has hold the correct value for this star).

```
SQLquery="update star set colour="+colour+",period="+period+" where  
starid="+starid;
```

Java Class: Pointing

Functionalities: It reads data from the source file with extension “.match” and inserts the data set into the table **Pointing**.

SQL statement:

1. Retrieve the relevant *exposureid* for current **Pointing** event from the table **Exposure**.

```
SQLquery="select exposureid from exposure where run="+run+" and  
colour="+colour+" and field='ngb'+field+" and ccd="+ccd+" and  
cameraid="+cameraid;
```

2. This inserts the **Pointing** dataset into the table based on retrieved *exposureid* value.

```
SQLquery="insert into pointing (exposureid,shiftx,shifty,a,b,c,d) values  
("+exposureid+", "+shiftx+", "+shifty+", "+a+", "+b+", "+c+", "+d+");
```

Java Class: StarIntensity

Functionalities: This is to read time intensity data for stars and then loads them into the table **Starintensity**.

SQL statement:

1. Retrieve relevant starid value from the table **star**:

```
SQLquery="select starid from star where field='ngb'+field+" and ccd="+ccd+"  
and index="+index+" and cameraid="+cameraid;
```

2. Insert the data set into the table **Starintensity** based on retrieved starid value.

```
SQLQuery="insert into starintensity (starid,time,run,flux,error,n1,n2,n3,n4,n5)
values
("+starid+", "+time+", "+run+", "+flux+", "+error+", "+n1+", "+n2+", "+n3+", "+n4+",
"+n5+")";
```

Java Class: GeneralDAO

Functionalities: It is an interface class between Java classes and the database. All Java classes need to create a GeneralDAO object when they want to communicate with the database.

Java Class: main

Functionalities: it is the main interface of the program. It offers a menu with accepting inputs for user to loading different kinds of source files into the database.

Format of the Text Based Files and the loading methods

The formats of the text-based files are important because they determine the development of file reading methods of the data processing Java application.

MOA's telescope outputs the imaging data as FITS format. FITS is a standard astronomical image format. Plante (1997) stated FITS shows multi-dimensional, regularly-sampled array of measures images. MOA text files are prepared from the FITS header, they has particular sequence. For instance, the contents of the exposure file *B144-ngb16-blue-1.info* is shown by Figure 18. From this text file, we can see the exposure had produced imaging metadata. The exposure run number is B1466, observed field is in ngb16, camera filter is blue, CCD field is 1, start observing time is 2452134.1213147 and end observation time is 2452134.133623. The Figure 19 shows a calibration data file contents, it includes RA, DEC and other specific calibration figures related to the field ngb is 1 and CCD is 1.


```
# /export/data/GB/B1466-ngb16-blue-1.fit.Z
SIMPLE = T / FITS STANDARD
BITPIX = 16 / FITS BITS/PIXEL
NAXIS = 2 / NUMBER OF AXES
NAXIS1 = 2047 / NUMBER OF PIX IN 1ST AXIS
NAXIS2 = 4095 / NUMBER OF PIX IN 2ND AXIS
BSCALE = 1.000000E+00 / REAL = FILE*BSCALE + BZERO
BZERO = 3.276700E+04 / REAL = FILE*BSCALE + BZERO
CCDTEMP = -120.0 / Camera temperature deg C
FOCPOS = f6.25 / Focus position
OBSERVAT= Mount John / Observatory
OBSTEL = B and C Telescope / Observing telescope
LOGITUD = -170.467 / Longitude (Negative means EAST)
LATITUD = -43.983 / Latitude (Negative means South)
HEIGHT = 1029.0 / Height (metres)
JDSTART = 2452134.131470000 / JD of begining of exposure
JDEND = 2452134.133623000 / JD of end of exposure
CCDADDR = 1 / CCD number (address) in the array
OBJECT = ngb16 / Name of field object
FILTER = blue / <630nm(blue) or >630(red)
```

Figure18: Contents of the exposure file: B144-ngb16-blue-1.info

```
17:47:30.000 //RA
-34:15:00.00 //DEC
1
1
-3376.736539671498 //Special value: A
4126.358992331345 //B
0.580524809652903 //C
0.585242628812929 //D
0.328410398068884 //E
-0.381173305325939 //F
```

Figure 19: Calibration data file gb-1.par

(The designer appended Comments)

The task of Java application is to follow the file sequence and read the required information from the text files. Then the application loads the data records into relevant tables. However, there is a challenge while loading the data records into the database. As Paul (2005) stated “bulk loading is more efficient than single-row loading”. Therefore, the best way for loading data records is to make the individual data record becomes a data heap, and then the program loads the data heap into database. But in this database, the situation is particular. One of astronomy data feature is its extremely big data size. The Java container- stack container was chosen to store individual data record. But with bigger and bigger data records saved in the container, the container was “out of memory”. The reason is any kind of containers; even a Linked-List container has a limited capability on saving data. The big

number of astronomical data sets always exceeds the container's capability. Therefore, the method 'single row loading' was used to loading this kind of data file.

- Single row loading: inserting each data set into database as soon as it reads out from the data source file. For example, the **Intensity value** data source file includes a large size of data records, such as hundreds of thousands of data rows. Due to container capability reason, no container can afford such a huge data set. Therefore, each data record was inserted in the database as soon as it reads out from the data file. In addition, in the **Intensity value** source file, only some of the information should be inserted in the database.

```
# /extern/work3/bondi/diff/B1767-ngbl6-red-1.diff.fit
B1767 2452367.192286 1 207.189 546.543 8584.8 1161.55 -4.67408 88.8633 0 1 0 (A1)
B1767 2452367.192286 4 592.736 228.419 112659 12625 170.12 984.242 0 8 0
B1767 2452367.192286 5 637.466 320.567 1206.64 910.141 2.76965 70.9011 0 8 0
B1767 2452367.192286 9 795.591 991.942 -242.63 3288.91 -14.2826 250.542 0 9 0
(A2)
# /extern/work3/bondi/diff/B1772-ngbl6-red-1.diff.fit
B1772 2452373.155399 1 207.189 546.543 30135.2 2242.7 -18.948 115.304 0 1 0
B1772 2452373.155399 4 592.736 228.419 120136 12747.5 81.3277 677.927 0 8 0
B1772 2452373.155399 5 637.466 320.567 2193.92 1042.43 -3.85267 55.218 0 8 0
# /extern/work3/bondi/diff/B1778-ngbl6-red-1.diff.fit
B1778 2452377.0849305 1 207.189 546.543 21863.2 1112.91 -11.3533 88.1904 0 1 0
B1778 2452377.0849305 4 592.736 228.419 79515 11372.4 178.025 912.908 0 8 0
B1778 2452377.0849305 5 637.466 320.567 -2473.01 668.37 4.79447 53.8243 0 8 0
B1778 2452377.0849305 9 795.591 991.942 -17227.8 4728.92 -4.56925 380.698 0 9 0
```

Figure 20: The structure of the source file

Figure 20 is a simple example of **Intensity value** data source file. From it we can see, this kind of data file describes some stars intensity values among different time. The stars are in Field 16, CCD 1, observed by red filter and the exposure run numbers are B1767, B1772 and B1778. In the source file, the first column is the run number for the exposure. The second column is the exposure time. The 3rd column is the star's index value in the field 16. The 4th column and 5th column are star's reference location on the image, but this information is duplicated with the data source file **star location**. The 6th column is the flux value and the 7th column means error value. From the 8th column to 13th column are other measurement values for star intensity. The row started by the symbol “#” is the metadata for the following data. This metadata row separates the data into many blocks, and each block of data shows certain star's flux value at different time.

Based on the file sequence, a loop was used to read the useful information and to avoid loading redundant data in the database. The loop structure can be shown as follow.

```
//The author defined one block of data is "the data between the data line starts
by symbol hash", such as block A2 in the Figure 14
While (not end the file){
    If the cursor is in the first block (Area A2 on the Figure 18){
        The program gets current star's starID from the table Star and
then saves this starID in a container. (The reason for doing this is to
avoid searching of this starID from the table again). Lastly, the
program inserts the record into table StarIntensity
    If the cursor is not in the first block and is not metadata{
        The program inserts the star's record into the table StarIntensity.
        The starID value can get from the container.
    }
}
```

- Bulk loading. Some data source files only include a small number of data record. For example, each **exposure** (.info) source file only contains one data records. Therefore a static Vector container was used to record heaps of exposure data records, and then these bulk of data records were inserted into the database at one time. This method can reduce time to setting up connection between the application and the database. An example of code implementation using bulk inserting is shown below (This example only shows the loop structures instead of the real Java codes).

```

//Class exposure can read exposure data files and save the data record in the
vector container
Public class exposure {
    //the static vector container can save bulk of exposure data records
    Public static vector<String> allexposure;
    //the function can read individual exposure.info file and save the data
    //record in the container allexposure
    Public void readfile();
    //insert bulks of exposure's data records into the database
    Public void loadtodatabase();
}

//The class main to call the exposure class object
Public class main {
    public static void main(String args[]) {
        loop (reading data files from B1466-ngb16-red-1.info to
            B1500-ngb16-red-1.info )
        {
            exposure temp; //produce an exposure object
            temp.readfile();//reading the data record and pushing the record
                //into the static container
        }
    }
}
//insert contains data records into database (bulk loading)
exposure:loadtodatabase();

```

4.5 Enterprise application design

4.5.1 Application architecture design

This section deals with the application design. Previously the website requirements were described. The hierarchy website structure can be shown as follows.

- 0. MOA welcome page
- 1. Log in main page
 - 1.1 Log in
 - 1.1.1 Modify member's details
 - 1.2 Register as new member
 - 1.3 Member to retrieve their password when they forget.
- 2. Gravitational microlensing introduction
- 3. MOA publications download
- 4. MOA observation equipments introduction
- 5. Astronomy data accessing
 - 5.1 Query data by general option
 - 5.1.1 Fills SQL command based on template query boxes
 - 5.1.1.1 Get query results by table on the web page
 - 5.1.1.2 Get query results in a data file and download it
 - 5.2 Query data by professional option
 - 5.2.1 Finish SQL query by user.
 - 5.2.1.1 Get query results by table on the web page
 - 5.2.1.2 Get query results in a data file and download it
 - 5.3 See stars' light curve by picture based on their time series intensity measurement value
- 6. About MOA
- 7. Discussion Board
- 8. Log out

In order to cover all these requirements and ensure that the application has a good application has a good architecture for implementation and maintaining, the MVC pattern was used. A Servlet was used as server-side processing to implement the controller layer, Java Beans to realize the model layer, and JSPs to fulfil the presentation layer. Figure 21 shows the application architecture. The message transformation rules are shown as follows:

1. Clients make a request through HTML forms to the Servlet Controller.
2. The Controller receives the request. Then it instantiates appropriate business object based on Java bean and calls the object method to perform business tasks.

3. The Java bean (model layer) contains business logic. It can receive parameters from the controller and perform Java bean functions. Finally, a representation of what will be displayed for the view layer will send back to the Controller by the Java bean.
4. The Java bean uses Data Access Object (DAO) to communicate with the database when the business task requires.
5. The Controller receives results from Java Bean and forwards request to the appropriate JSPs
6. The JSPs receives process results from Java bean and the controller. Then it formats the “return page” and shows it to the clients.

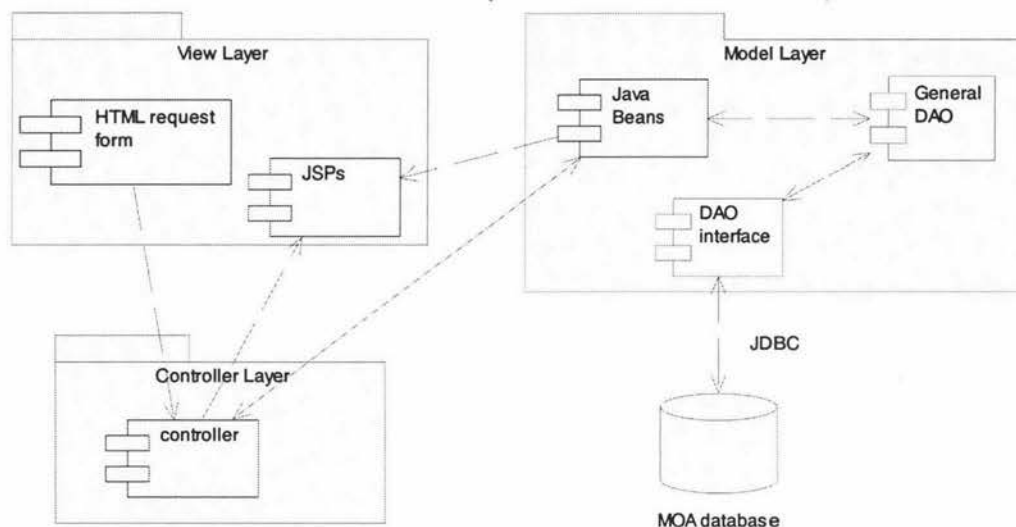


Figure 21: The simple component diagram for the application

In addition the application needs to have two person roles, including general member and administrator. Administrator has the right to organize the registered members and to maintain the discussion board. General members have privileges to access MOA astronomical data, but they only can read this data.

Based on application architecture, the design is divided into four parts, including *distribution requests design (controller layer)*, *presentation logic design (view layer)*, *business logic design (model layer)*, and *data access object design*.

4.5.2 Servlet Controller design (Controller layer)

In the MVC structure, the controller is responsible for the coordinating the data flow between the model and view layer, responding to user requests, and management of data in models with actions. So if the controller is written properly, it will channel request data and invoke action calls for any type and number of views. This project used a Servlet as controller for the application. The definition of the Servlet in XML deployment descriptor file is shown as follows:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE web-app (View Source for full doctype...)>
<web-app>
  <servlet>
    <servlet-name>controller</servlet-name>
    <servlet-class>mypackage.controller</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>controller</servlet-name>
    <url-pattern>/controller</url-pattern>
  </servlet-mapping>
  ....
</web-app>
```

In order to let the controller know what action to perform and JSPs to display next, it was decided to put action keys into JSP form as hidden variable indicating to the controller what to do next. When the *doPost* method in the controller receives request from JSPs, a Reflection API was taken to instantiate the Action class (The action class are used by the controller to perform any sort of action on the user's behalf). For instance, the controller checks the hidden variable value and decides what action class will do next. Normally, in the action class, the controller receives the other parameters from JSPs and populates necessary business objects based on Java bean class. Then, the controller calls the object method to perform business task. And the object method will return a JSP view back to the controller. Lastly, the controller forwards the page to view layer.

4.5.3 Presentation logic design (View layer)

This section describes the design of the presentation logic for the application. Haque and O'Connor (2002) stated the presentation logic is the code that dynamically generates display elements and programmatically decides what content is displayed to the user. Therefore, the purpose of the presentation logic design is to show web contents in a formative way.

In this website, JavaServer Pages (JSPs) were used to implement presentation logic. As an important J2EE component, JSP is an extension of Java Servlet technology. Actually, JSP is translated into Servlet source code automatically when it is deployed on the server. It allows Java code to be embedded in HTML (Dustin, 2001). So, with JSP, the web developer can use HTML to display static page contents and Java code to show dynamic page contents. Moreover, because JSP embedded with Java code, it can take all the advantages, which Java has. However, while JSP technology makes web page contents become flexible, it also makes pages code become mess. For example, mixed HTML and Java code cause them significantly less readable and debugging hard. Therefore, how to combine HTML and Java code is a challenge for the JSP technology. The JSPs were designed with the following design principles in mind.

1. Maximally separate HTML from Java code. If HTML and Java code are separated, the designer can concentrate on one part at different time. For example, in most of JSPs, HTML codes are put together at the top of the page and java code was at the bottom of the pages.
2. Place business logic in Java bean. When Java codes are included inside of the JSPs, they are not accessible by other JSPs. In contrast, Java bean can be used by all of the JSPs even by other applications in the application. Therefore, in order to enhance the code reusability, the business logics were put in Java bean.
3. Use *include* directive. The mechanism supports JSPs includes the content of a specified file in the JSP. Thus, the *include* mechanisms can reduce code redundancy and promote reusability. For example, the text information for the web title and menu needs to be included in all JSPs. Therefore, a JSP which

contained the website title and the menu information was created. And then it was included in the required JSPs.

4. Use style sheets. Style sheet files enable developers to control the layout of the web pages. This project used Cascading Style Sheets (CSS) to control font families, font size and table characteristics. Style sheet allows the developer to make changes in on location and those changes reflect on all related pages (Dustin, 2001). As a result, it can increase the maintainability and consistent appearance to user.
5. Use JSP exception mechanism. Exception's stack trace is important information for both of developers and users. Therefore, a JSP called *errorpage* was used to catch all the exceptions, which are thrown by JSPs.

The JSPs were divided into three types according to their functionality. Firstly, it is static page and it only contains static content. For example, the web page Telescope.jsp introduces MOA's observation equipments. It can be shown to clients directly without change. The second type is "post page". This kind of page includes HTML forms. It posts request to the Servlet controller and to ask dynamic information. The last type is "answer page". It receives the display data from the Servlet controller and shows them in a formative way to clients. The following description gives details of web pages with their functionalities and types. The whole implementation of the application can be seen in the enclosed CD.

Page Name: Header.jsp

Page Type: header and static

Functionalities: This page creates a header for web site. And it also includes a menu, which can link to other JSP pages.

Page Name: Index.html

Page Type: Post

Functionalities: This is an entrance page for the web site. The page contains a form to allow users to enter their user name and password as login details. This form posts the input values into the controller to verify the user.

Page Name: Index2.jsp

Page Type: Answer

Functionalities: This shows login result. If user has a correct login, it shows welcome letter. Otherwise, it tells user to login again. If administrator logs in, the page supplies links to the web pages for maintaining the web site contents.

Page Name: Register.jsp

Page Type: Post

Functionalities: This Page allows a new user to register. User should input their details such as first name, Email address and student ID. When users submit the details in the HTML form, the details are posted to the controller. Then the controller creates a process object based on Java class registerBean. This new object handles the registration process.

Page Name: Modifydetails.jsp

Page Type: Post

Functionalities: This provides users to modify their details.

Page Name: result_register.jsp

Page Type: Answer

Functionalities: The page receives parameters from the controller. And then, according the parameters value, it displays user' registration and modification process result.

Page Name: servlet/sendPassword

Page Type: Post

Functionalities: This is written as Servlet, it supports to send users' password to their email box when users forget their password.

Page Name: Maintain.jsp

Page Type: Post Answer

Functionalities: administrator uses it only. This is used for organization the whole registered members. If the numbers of members is more than 15, the page shows the member details in separated pages.

Page Name: Data.jsp

Page Type: Static

Functionalities: This is entry page for the astronomy data fetching. It offers two links. One link is to the page dataGeneral.jsp. It allows user to complete SQL query based on pre-prepared conditions. Another link is to page dataProf.jsp. User can input SQL query by own to query data. And then the application will use the produced query to fetch data from the database.

Page Name: dataGeneral.jsp

Page Type: Post

Functionalities: The page supports user through general operations to query data from database especially for querying star time intensity values. User can input conditions in the text boxes to decide SQL query command for fetching data. Then the page sends the parameters to the controller through HTML form. If the user leaves the text box empty, it means user does not specify this condition. The specific text boxes included star identifier, duration of the intensity value, RA, DEC and distance.

Page Name: dataProf.jsp

Page Type: Post

Functionalities: professional users use this page. The users need to input SQL command by them in order to get image metadata and star time intensity value. The Java bean will retrieve the data from the database after a general syntax checking for the input query. But only “select” command is allowed. And then the page delivers user’s query to the controller.

Page Name: viewData.jsp

Page Type: Answer

Functionalities: The page gets data from the controller and shows them within table. If the number of data is over 25 rows, the page divides the data into several pages.

Page Name: viewFile.jsp

Page Type: Answer

Functionalities: The page supports users to download their astronomy data in a data file. The downloaded file is produced by Java bean and is saved in the folder

“URL:temp_file/student id” on the server, and it will be deleted when the user logs out. The Java class downbean.java is used for transmit the data file from the server side to the client side.

Page Name: lightcurve.jsp

Page Type: Post

Functionalities: This allows user to see star's light curve in the pictures according to time series intensity data. Users need to point out the identifier of the star and displaying format, and then the page sends all parameters to the controller. The controller receives parameters from this page. And then it produces SQL command for querying data from database. Then the controller produces star's light curve by picture through an application 'Gnuplot'. Lastly, the page 'showlightcurve.jsp' displays the picture to the user.

Page Name: showlightcurve.jsp

Page Type: Answer

Functionalities: The page shows stars' light curve by picture. Only logged in user has right to see star's light curve. Firstly, the page receives parameters from the controller. The parameters include light curve picture's file name and physical location for the image file. The page also includes two buttons called 'Next Star' and 'Previous Star' in the HTML form which can show the next star and previous star's light curve. The controller is able to produce star's light curve by picture and save the file on the server.

Page Name: Addthread.jsp

Page Type: Post

Functionalities: Users can input subject details on the discussion board and the data is posted into the controller. But only login users are able to post subject.

Page Name: discussion.jsp

Page Type: Answer

Functionalities: This supports a discussion board for users to discuss and to share their ideas. This page only shows thread subject information and the page discussiondetails.jsp shows the reply messages for each subject. The page shows all

subjects by last-post-time order and there are up to 15 threads in one page. More threads are shown in separated pages. When normal users come to this page, they can read subjects on the board. If the administrator comes, he or she can delete any threads from the discussion board. The deleting operation can be done by the controller through the Java bean *discussionBean.java*. The Java bean passes the operation results back to this page through controller.

Page Name: Replymessage.jsp

Page Type: Post

Functionalities: Users can reply a subject on the discussion board in this page. It will post message to the controller. But only login users can post messages.

Page Name: discussionDetails.jsp

Page Type: Answer

Functionalities: The page shows the reply messages for a particular subject in the forum. And only registered users are able to post messages. It can receive parameters 'id' from the page discussion.jsp which indicates the subject details.

Page Name: Gravite.jsp

Page Type: Static

Functionalities: The page simply describes gravitational microlensing concept and shows how to use this technology to observe stars. The content in this page is static and retrieved from MOA Auckland university website.

Page Name: Telescope.jsp

Page Type: Static

Functionalities: The page introduces the new MOA telescope details and provides some pictures for it. The information on this page is from MOA Auckland university website.

Page Name: about.jsp

Page Type: Static

Functionalities: This offers a brief introduction of MOA group.

Page Name: publication.jsp

Page Type: Static

Functionalities: This shows some of the MOA's publications and supports links to download these documents. The description of publication is copied from MOA Auckland university website.

Page Name: ErrorPage.jsp

Page Type: Answer

Functionalities: This is the error page used by all of the JSP pages. It simply displays any exceptions that are thrown within the JSP pages.

Page Name: Logout.jsp

Page Type: Answer

Functionalities: This page simply ends the current session bean and displays a link back to the page index.html, so that the other user can log in. Furthermore, it can delete all the data files which are produced by the user during transaction and leave on the server.

Page Name: Formating.css

Page Type: Format file

Functionalities: This sets the whole web page style.

4.5.4 Business logic design (Java Bean Model layer)

Now, let's move to business logic design. The purpose of the business logic is to perform business-related tasks and to communicate with the database. The business-related tasks in this application include user registration, querying astronomical data with or without data mining knowledge and organizing the discussion board. However, this thesis did not mention more details about putting data mining knowledge in the application. It is a limitation. But the designer might do further research of data mining knowledge in the Java bean in the future. Under the MVC, when the controller executes an action method, the action may populate a Java bean to perform business tasks. Therefore, Java beans are able to instance field, get

parameters, store data and contain methods to perform business tasks. In this project, eight Java beans were created to perform the business tasks.

Table 5 states the designed Java beans with details and functionalities.

Table 5: Java Beans in the application

Bean name	Description
userBean.java	It is active when users log in and it is inactive when users log off. Generally it saves user's details, such as user name, student ID and email address. This information is used by some web pages. And one important function of userBean is it can delete the useless files from the server when the users log off. These useless files are created when users do operation on the website and left tem on the server side, such as light curve image files and downloaded data source files.
indexBean.java	It accepts users' student ID and password as parameters from the controller. And it can create a data access object based on class GeneralDAO. The object can connect to database and check whether users have a correct login or not. If yes, the object retrieves other properties, including user email address and user name from the database, and then it returns the login result to the web pages.
registerBean.java	It accepts parameters from the controller and register new member and then it saves user's information in the database. It also supports members to modify their details in database.
dataBean.java	It is able to handle the operations between users and astronomy data. The main functions of this bean include building SQL command based on people's request, fetching data from the database and saving the fetched data in data files.
discussionBean.java	The bean is used for discussion board. It supports users to add new threads and delete thread from the board. In addition, it is able to get all thread subjects information from the database and pass the results to JSP for displaying.
disdetailsbean.java	It has similar functions with discussionBean.java. It can post reply, delete replies message and display all replies for a particular subject in the forum.
imageBean.java	It is used for producing star's light curve by pictures. Before calling this bean, the application needs to call the function <i>dataBean.java.makefile()</i> to produce data source file which supports the source data on plotting in the application Gnuplot. This bean can produce a PLT file (it contains pipeline of Gnuplot commands) and then it runs the file on Gnuplot to produce the image files. Then the images can be shown by JSP. The generated images are saved in the folder <u>URL:temp_file\</u> +studentID.
maintainBean.java	It is used for charging member's information. In particular, it supports functions for deleting members from the database

The General Java Classes for supporting business tasks are shown by Table 6.

Table 6: General Java classes in application

Class name	Description
generalDAO.java	It is used for controlling the interaction between the database and the application. It not only can produce SQL query, but also can execute query in the database. Moreover, it can return the query results.
MailService.java	It handles sending emails to users.(this class is not my own and it is adapted from the J2EE and beyond book by Art Taylor, 2002)
runningDownloader.java	It provides download files from server side to client side.
constants. java	It contains a list of integers with names to make query's running are more readable.

4.5.5 Data access design

In applications most business logics are related to database systems. Therefore, how the Java beans communicate with database is an important part of the design. If the Java beans connect and disconnect with database within an appropriated time, the application performance will be good. Otherwise, the performance will suffer. Therefore, Data Access Object (DAO) was used to control the communication between the application and the database. Roman (1997) said DAO is a component which provides a common interface between the application and one or more data storage devices, such as database or file. As a core of J2EE design pattern, the advantages of DAO are that business objects no longer require knowing the final destination for the information it manipulates. As a result, the design activities can be divided into two parts. In part one, the design focuses on realization business logic in the Java beans, and in part two the design concentrates on how the beans connect with the database. Figure 22 shows a model that using DAO to connect with database.

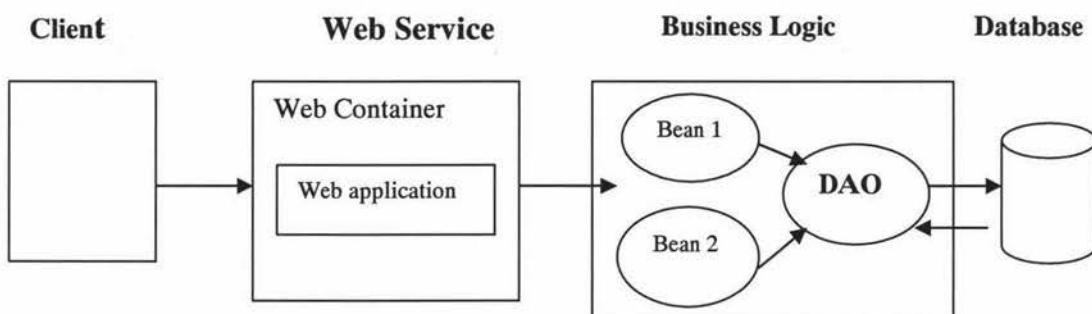


Figure 22: J2EE application Model using DAO to communicate with Database

4.5.6 An example of passing messages among different layers in MVC pattern

Some simple examples are given that show how the messages go through among different layers. The example is user login process.

View layer (Post view):

Index.html: This has an HTML form with a submit action pointing to the controller Servlet. The hidden parameter *signal* guides the controller what to do next.

```
<form method="post" action="controller" name="input">
  <input type="text" name="studentID" size="25">
  <input type="password" name="password" size="25">
  <input type="hidden" name="signal" value="index">
  <td width="82" height="24"><input type="submit" name="submit"
value="Sign In" onClick="return validate()"> </td>
</form>
```

Controller layer

Controller.java: This receives requests from JSPs. Then, the controller calls action class including Java bean methods based on passed parameters. The Java bean object will return a JSP to the controller and then the controller redirects the request to this page.

```
public class controller extends HttpServlet {
protected void processRequest(HttpServletRequest request, HttpServletResponse
response)
    throws ServletException, IOException {
String signal=(String)(request.getParameter("signal")); //get parameter signal
if(signal.equals("index")){ //the request is from the page index.html
    studID=request.getParameter("studentID"); //get parameters
    passwd=request.getParameter("password");
    indexBean temp=new indexBean(); //create an object based on Java bean
    temp.setStudentID(studID); //pass parameters to object
    temp.setPassword(passwd);
    nextpage=temp.checkStudentPassword(request,response);
    //call object method and get return page
    RequestDispatcher //redirect to the return page
    dispatcher=getServletContext().getRequestDispatcher(nextpage);
    dispatcher.forward(request,response);
} else if (signal.equals("register")){ //for other requests
.....
}
```

Model layer

indexBean.java: it uses for testing whether users have a correct login or not (business logic). It creates a DAO object to communicate with the database. Lastly, it returns redirect pages back to the controller and saves procession results in the variable *request*.

```
public class indexBean extends Object implements
constants.java.io.Serializable {
    public String checkStudentPassword(HttpServletRequest request,
        HttpServletResponse response) throws Exception {
        generalDAO GetData = new generalDAO();//create a DAO object
        String login="";
        try{
            //get relevant data from database
            ResultSet myResult
            =GetData.runMyQuery(CHECKPASSWORD,studentID,password);
            int numofrows = 0;
            while(myResult.next()){
                numofrows++;//count how many rows were returned
                email = myResult.getString("st_email");//set email address
                firstName = myResult.getString("st_fname");//set first name
                lastName = myResult.getString("st_lname");//set last name
            }
            if(numofrows == 1) login="true";
            else login="false";
        }catch(Exception E){//catch the exception
        }finally{
            GetData.Closeconnection(); //close current connection
        }
        //set parameters for the display JSP
        request.setAttribute("login", login);
        request.setAttribute("firstName",firstName);
        request.setAttribute("lastName",lastName);
        return "/index2.jsp";
    }
}
```

DAO class

Data Access Object (DAO) controls the communication between application and database. The following codes show the Java bean *IndexBean.java* matched function in the class *GeneralDAO.java*.

```

public class generalDAO implements constants {
    public ResultSet runMyQuery(int whichQuery,String SQLextra1,String
SQLextra2,String SQLextra3 ,String SQLextra4,String SQLextra5) throws
Exception {
        //Depending on whichQuery variable value, the method creates the SQL
statement based on the other passed parameters
        if(whichQuery ==CHECKPASSWORD){
            SQLquery = "SELECT st_email,st_password ,st_lname, st_fname FROM
student where st_studid = '"+ SQLextra1 +"'";//create sql statement to find
matching student record
            SQLquery = SQLquery + " AND   st_password =" + SQLextra2;
        }
        try{
            //Loading JDBC Driver and set up connection
            Class.forName("org.postgresql.Driver").newInstance();
            theConnection =
            DriverManager.getConnection("jdbc:postgresql:member","Terry","11160
608");
            theStatement=theConnection.createStatement( ..);
            //return the result set to the calling object
            return theStatement.executeQuery(SQLquery);
        }catch (SQLException E) {
            //catch any SQL exceptions and throws them up to
            .....
        }
    }
}

```

The page interactions

The web application interaction is shown on Figure 23.

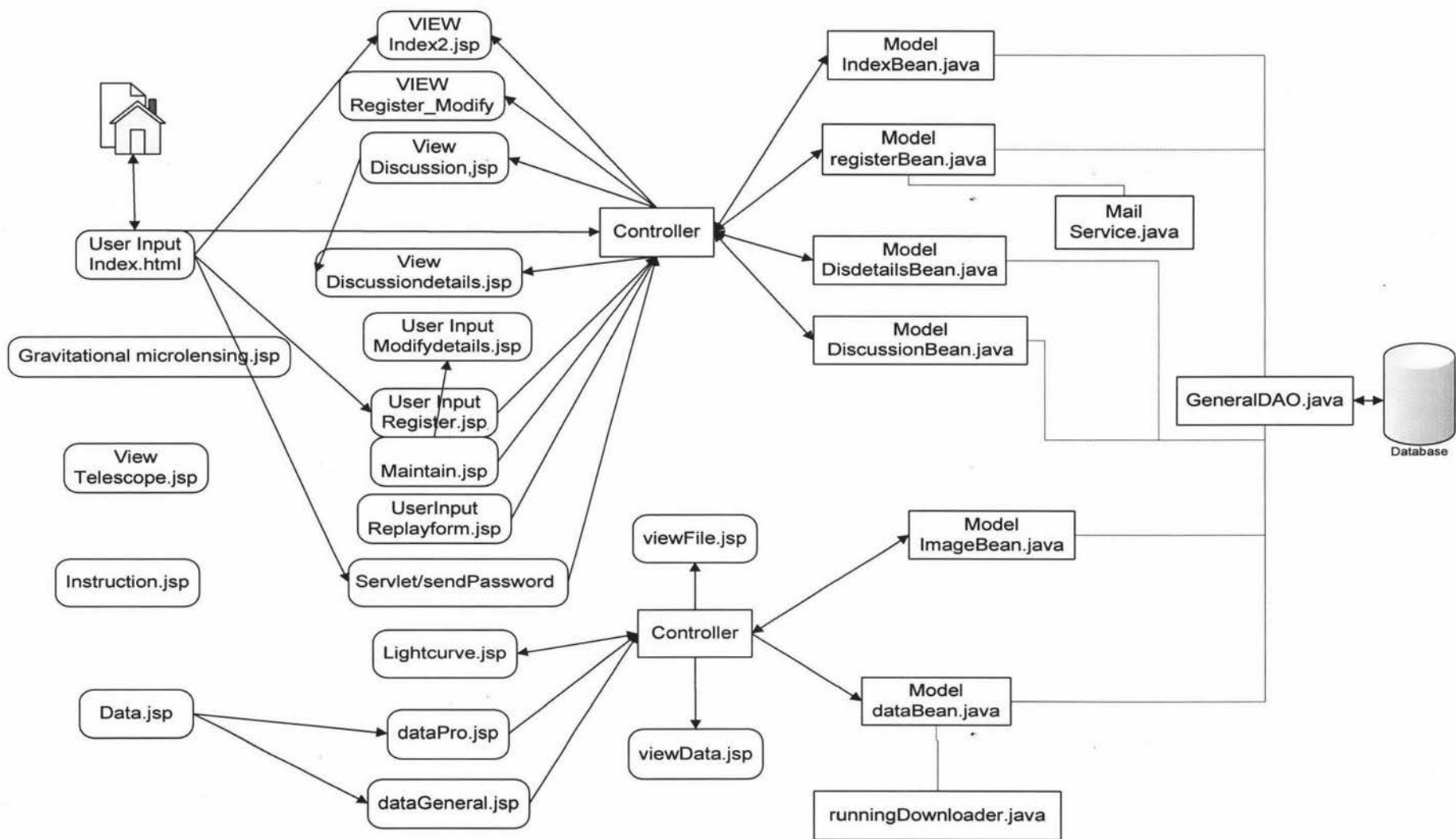


Figure 23: Pages interactions

4.5.7 Techniques used in the application

1. Pagination in JSP

In websites, it is common to paginate a large number of records which are retrieved from the database in JSPs. This situation happens in the page *viewData.jsp*, *discussion.jsp* and *discussiondetails.jsp*. several lines of Java codes were written to implement pagination. PostgreSQL supports the query feature called *offset* and *limit* clause to help realize pagination. The *limit* clause is able to limit the number of rows returned. And *offset* stands for skipping some of rows before beginning to return rows. This feature can make the pagination become easier, because SQL command can query needed rows from the database and show them on the page. Sample codes are shown below.

```
SQL command: select * from table limited 25 offset intPage*25
//intPage is the display page index. The command can query 25 rows from
the table and the row starting from the number intPage*25.

//In JSP, the variable page shows which page should be displayed
page=Request.getparametre(page);
ResultSet sqlRst; //the result set sqlRst holding the display data.
//return how many columns in the result set
int numColumn=sqlRst.getMetaData().getColumnCount();

//The below codes print the column title
for (int j=1;j<=numColumn;j++){
    out.print(sqlRst.getMetaData().getColumnName(j).toUpperCase());
}

while (!sqlRst.isAfterLast()){ //print all result set data
    for (int j=1;j<=numColumn;j++){ //print each row's attribute
        out.print(sqlRst.getString(j));
    }
    sqlRst.next();
}

//below codes are used for changing the page index under HTML
<a href="discussion.jsp?pages=<%=Page+1%>"> next page</a>
//go to next page
<a href="discussion.jsp?pages=<%=Page-1%>"> Previous page</a>
//go back to previous page
```


In the Java program, a variable called `intPageSize` was designed to limits how many rows can be displayed in one page. At the moment, the value of `intPageSize` is 25, but it can be changed freely.

However, this method produces more connections between the database and the interface, because it makes connection with the database in every page. Another pagination method was used. It is to query the entire datasets from the database and then paginate them in the JSPs (the dataset as a session bean in the JSPs). But under this method, the server has to record a big dataset during pagination. Therefore, a suitable method was chosen based on dataset size. If the dataset has a really big size, the first method was used to do pagination. In contrast, if the dataset size is not big, the second method was taken to paginate the data.

2. Passing parameters

Some parameters are transferred from one page to another page. In general, the website used two methods to complete this.

The first method is to pass parameters from view layer to the controller layer through HTML form by URL encoding. Sample codes are shown below.

```
<form name="message" method="post" action="controller" >
//this form is used for passing form parameters to the controller
<input type="text" name="studentID" size="25">
//the parameter studentID with user input value will pass to controller
```

Retrieve the parameter in controller:

```
String name=request.getParameter("studentID");
```

The second way is to pass parameters by HTML code.

```
discussiondetails.jsp?id=10 //the parameter has not been encoded, so it can be
seen by client.
```

Retrieve the parameter:

```
request.getParameter("id")
```

In addition, when Java bean used as session scope, it can be used as passing parameters as well, more details please see the section introduction of Java bean

3. Connection with database

The below codes not only show how the application creates a connection between PostgreSQL database and the application, but also show how the application runs the SQL. Consider an active client under PostgreSQL with user name "terry" and the password is "12345678".

```
try{
    Class.forName("org.postgresql.Driver").newInstance();
    //connect with POSTGreSQL
    //Loading JDBC Driver
    theConnection =
    DriverManager.getConnection("jdbc:postgresql:ass1","Terry","1234
5678");
    //set up the user name and password for database connection
    theStatement=theConnection.createStatement();
    theResult=theStatement.executeQuery(SQLquery);
    theResult.close();    //close the connection and other variables
    theStatement.close();
    theConnection.close();
}
```

4. Java Beans

Java beans handle the business logic in the application. Some of Java bean technologies used in the web site are shown below.

The tag is used to declare and initialize the Java bean class in JSPs with page scope

```
<jsp:useBean id="userBean" scope="session" class="myPackage.userBean"
/>
```

The tag is used to set the value of all the properties in JSPs

```
<jsp:setProperty name="userBean" property="*" />
```

The codes create a business object based on Java bean.

```
Create an object based on userBean:
userBean temp=new userBean();
```

Every Java bean object needs to have its scope in JSPs. Scope refers to the lifetime of the object stays in memory. This web site used *page* and *session* Java bean in JSPs.

Page scope means the Java bean is created when user starts to use the page and it is destroyed when user leaves the page. In other words, the Java bean only maintains its information in the single page. An example of Java bean with page scope is *dataBean* on page *viewData.jsp*, it shows astronomy data from the database. The bean needs to be destroyed and then released memory when user leaves the page, because the bean's information is not used for other pages any more. When Java bean's scope is *session*, it means the Java Bean will keep active among the pages. Every visitor visiting the page will have a separate session bean. And the visitor can retrieve session Java bean's data when they need. However, too many session beans will increase the server's workload, because the server has to record the session bean's information in the memory for each client. The website used a Java session bean which is called *userBean* to record user's details. Once user logs in, this small bean will be active. And *userBean*'s information such as user's name and Email is used in most JSPs. The tag for using the *userBean* is shown below.

```
<jsp:useBean id="userBean" scope="session" class="myPackage.userBean" />
```

The following codes state how to call bean's functions in JSPs.
Calling bean's functions in Java script:

```
userBean.setFname();
```

Calling bean's function in HTML code:

```
<jsp:getProperty name="userBean" property="message" />
```

5. Using http protocol to download files

The application need to support users to download the astronomical data files from the server, so a Java class called *runningDownloader.java* was created to finish the task. The simple codes are show below.

```

response.setContentType("application/x-download");
// set the content type of response

response.addHeader("Content-Disposition", "attachment;filename=" +
filenamedisplay); //add response header
try
{
    //return a Servlet output stream which is suitable for writing binary code
    in response. The reason for do this is the Servlet can not encode binary
    data.
    output = response.getOutputStream();
    fis = new FileInputStream(filenamedownload); //open the source file
    byte[] b = new byte[1024];
    int i = 0;
    while((i = fis.read(b)) > 0)
        // the loop for reading from source file and write data to download file
        {
            output.write(b, 0, i);
        }
    output.flush(); //commit the response
}

```

6. Discussion board

Figure 24 shows the physical table structure in the database for saving the discussion board data. From the Figure we can see all subjects are saved in the table *discussion* and each row in the table present one subject in the forum. The recorded information includes each subject's title, author, when it was posted and the contents of this subject. The table *disdetails* records each subject's reply messages and each row stands one reply message. The foreign key *id* is used to links subjects and their reply messages.

```

CREATE TABLE discussion (
    id serial primary key,
    title char (30) NOT NULL
    default "",
    author char (20) NOT
    NULL default "",
    time timestamp,
    content text
);

```

```

CREATE TABLE disdetails (
    detailsid serial primary key,
    id int,
    author char (30),
    time timestamp,
    content text,
    foreign key id references discussion
    on delete cascade on update cascade
);

```

Figure 24: The table structure for saving discussion board data

The simple queries for retrieving data are shown below.

The query for gets all subjects information which is posted after Sep 20, 2006 and lists the latest thread on the top.

```
Select * from discussion where time>'20-09-2006' order by time
```

Retrieving a particular thread from board and appending the latest reply on the top.

For example, the queries for getting the subject's ID=10 and its replay message are:

```
Select * from discussion where id=10; //getting message subject information  
Select * from disdetails where id=10 order by time; //getting its replay messages
```

7. Class Inheritance

In the application, the class *constants* is a parent class of some classes (Figure 25). *Constants* class contains a list of integers with names to make database query more readable. Therefore, any classes, as long as they have communication with the database, will be a children class of *constants*.

The codes for using inheritance are show below.

```
//Implementation of constants class (parent class):
```

```
public constants {  
    int MODIFYUSER=1;  
    int REGISTERUSER = 2;  
    int GETPASSWORD = 3;  
    int RETRIEVEDATA = 4;  
}
```

```
//Declaration of IndexBean class (child class)
```

```
public class IndexBean extends Object implements  
constants ,java.io.Serializable {}
```

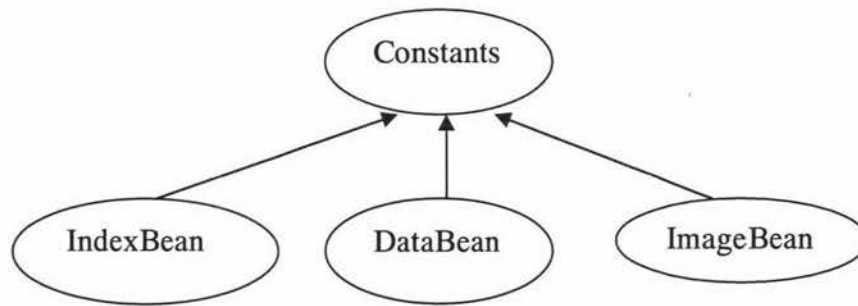



Figure 25: Structure of inheritance relationship

8. Get current path for the application file system

Based on HTTP protocol, the Java code for getting current physical path of the website is:

```
String path=request.getRealPath("");
```

//the code is used for finding the current physical path of the web application, but it doesn't work when the web application is compressed.

9. Show star light curves through Gnuplot

In the website, the web page *lightcurve.jsp* shows star light curves in pictures. User can choose any stars within any period to produce light curves. So the light curves must be produced dynamically. This application used external application, which is called Gnuplot to produce light curve. Gnuplot is a portable command-line driven interactive data and function plotting utility (Gnuplot, 2006). It allows users to visualize mathematical functions and data. Gnuplot supports a plot control file with a ".plt" extension and users can put all of their Gnuplot commands in that control file. In order to dynamically produce light curves, four steps were used to fulfil this function.

(1) Producing data source file.

A data source file was created on the server to store user required star time intensity data. The source file is saved in the folder "url:temp_file"+ studentID. It insures each user has a unique folder for saving their files on the server.

```

theResult=GetData.runMyQuery(GETASTRONOMYDATA,SQLque
ry);
//Getting time intensity data from the database
File folder=new File(path+"\\temp_file"+"\\"+studentID);
if (!folder.exists())folder.mkdir(); //create folder for file
filelabel=studentID;//the source file name start by userID
filename=filelabel+ ".txt";
//change the file name as random number
//because one user may request several pictures
File myFilePath=new File(folder,filename);
myFilePath.createNewFile();//create source file
FileWriter resultFile=new FileWriter(myFilePath);
PrintWriter myFile=new PrintWriter(resultFile);
while (theResult.next()){
    tempString=theResult.getString("time");
    int j;
    for (j=1;j<numColumn;j++){
        myFile.print(theResult.getString(j)+ "
    }
}

```

(2) Producing the controlling file for Gnuplot.

The controlling file includes pipeline of Gnuplot commands. This file sets the data source file, the output terminals type and format of output pictures. An example of control file with contents is shown by next.

```

//control.plt
set terminal gif small size 640,480
set size 1,1
set xlabel 'time'
set ylabel 'flux'
set output
'MOA_web_site\build\web\temp_file\98765432\3498765432.gif'
plot 'MOA_web_site\build\web\temp_file\98765432\98765432.txt'
using 7:10

```

(3) Executing Gnuplot with control files under Java bean to produce picture.

The syntax for calling an executable program under Java code is shown below.


```
Runtime r=Runtime.getRuntime(); //create a thread
Process p=null;//initialize the thread
p= r.exec("\\pgnuplot.exe "+path+"\\ "+control file);
//run the GNUPLOT application with control file
int a= p.waitFor();
//the program continues after current thread is finished
```

- (4) Showing the produced pictures to users.

After the image is produced, it is saved on the server, the following codes showing the image under HTML code.

(The variable image_file_name contains the physical file path and file name of the light curve).

```

```

- (5) Delete the produced file.

After the picture has shown to user, the picture files should be deleted from the server.

10. Using DAO to communicate with database

In web application, there are quit a lot of communicates with the database. The Data Transfer Objects (DTO) called general DAO was created to pass data between users and the database. The protocol type for creating a DTO and using its function are shown below.

```
generalDAO GetData = new generalDAO(); //create a DAO object
try{
    theResult=GetData.runMyQuery(modifyuser,SQLquery); //DAO
    object take responsibility for working with database and return the
    query results
    GetData.closeconnection(); //close the connection
}
```

11. Using table view instead of real table

View can join many tables together and show their data relations. Therefore, when the web application queries data, there are advantages for using view instead of the real table structure. For example, using view can simplify SQL command (users do not

need to know the complexity real table structure) and improve security (users cannot know what the real table structure is). In this web site, the web pages *dataProf.jsp* and *dataGeneral.jsp* are used for querying astronomy data from database. The view *view_star* helps clients to fetch data.

4.5.8 Maintaining The Application

Maintaining the application is an important stage for the application life. The application designed a special role which is called administrator for maintaining the website.

The administrator duties include:

- Administrate the website members

More and more members may register as members. How to manage these members will be a question for MOA. The application offers particular web pages for administrator to organize the members, such as deleting and adding members in database.

- Organize the discussion board

In the discussion board, any member can post threads in the forum. The administrator has right to delete any threads from the forum.

Figure 26 shows how administrator organizes the discussion board. From the Figure we can see, when the administrator logs into the web page *Discussion.jsp*, he or she has privilege to delete any threads from the forum.

MOA-Microlensing Observations in Astrophysics

Currently logged In as : terry Xu 11111111

Home

Gravitational

Publications

MOA Telescope

Captured data

About MOA

Discussion Board

Logout

Copyright © 2006
Massey University

Title	Author	Time
aaaaaaaaaaaaaaaaaaaaaaaa	cc	2006-10-30 16:03:13.64
cccccc	cc	2006-10-29 16:11:20.828
ssssss	cc	2006-10-29 15:18:45.328
look here	terry	2006-10-29 15:08:33.515
ssssss	dd	2006-10-28 00:00:32.234
ssssss	dd	2006-10-28 00:00:27.171
ssssss	ffsd	2006-10-27 22:24:18.406
ssssss	ffsd	2006-10-27 22:24:13.781
ssssss	dd	2006-10-26 17:25:48.375

1 of 1

Delete

Figure 26: Screen shot for Administrator controlling the forum

In addition, as time progresses, new astronomical data will be obtained. In order to update the database, the application needs to load new data into the database. The Java application namely *MOA_data_ingestion* can load new data into database. But the current application only offers limited functions for controlling the database; it should be improved in the future. For more details please see the section 4.4.

5.0 Project Deployment

5.1 Database deployment

5.1.1 MOA database creating

In this project, two databases were created for storing data. The database **MOAdata** stores all the astronomy data and the database **Member** saves the web site related data, such as forum data and member details. The two databases were created under PostgreSQL. The following steps introduce how to initialize database. (Notice this instruction is working well under PostgreSQL 8.1 with Windows operation system. For the other systems, it might have slightly different.)

1. Open PostgreSQL command line client
2. Using script file "createdatabase.sql" to create database, tables, indexes and views. This script file includes SQL commands to set up the database.

To run a script file issue: `\i createdatabase.sql`

Notice: the command `\cd +path` used to change the physical working directory.

If the script file is loaded correctly, the PostgreSQL will give positive feedbacks.

After the databases are created, the Java application **moa_data_ingestion** will be used to insert the astronomy data from source files to related tables.

5.1.2 Data Loading

A Java application called **moa_data_ingestion** was created for loading the data into the database. The application followed BluePrints format, because it guides an efficient way on the enterprise application. The Java BluePrints program defines the application programming model for end-to-end solutions using the Java EE platform (Sun Microsystems, 2006).

The requirements for running the application:

- JDBC library File: *postgresql-8.1-407.jdbc3.jar* (under Windows operation system)

- One active client on PostgreSQL:

Client name: *Terry*
Password: *11160608*

The user needs to run this application, and to follow the screen instructions to load the data. Normally the application asks users to input some parameters, such as loading file name, its location and the file type in order to find source files and the related tables. The enclosed CD contains some data source files under the folder: **CD disk\data**.

5.2 Enterprise application deployment

An enterprise application, namely **MOA_web_site** was created. And it followed BluePrints structure as well. The entire application was written under NetBean4.1. The application was designed to run under Tomcat application server. Moreover, it also can be run under other application servers. The easiest way for running the application is to open the application as a project under Netbean and run it use the bundled Tomcat server.

The requirements and notices for running the application:

1. JDBC library File: *postgresql-8.1-407.jdbc3.jar* (under Windows operation system)
2. mail.jar: this JAR file includes library for sending mails under JSP code.
3. activation.jar: This JAR file contains the classes that make up by JavaBeans Activation Framework.
4. The physical location for saving GNUPLOT application is \\gnuplot\\bin\\pgnuplot
5. The application is tested under Microsoft Windows system.
6. All Java classes can be compiled by JDK1.5 or above version.
7. Administrator Password and user name:

The default **Administrator ID**: 11111111
The default administrator **Password**: 1234

Figure 27 shows the application physical saving structure.

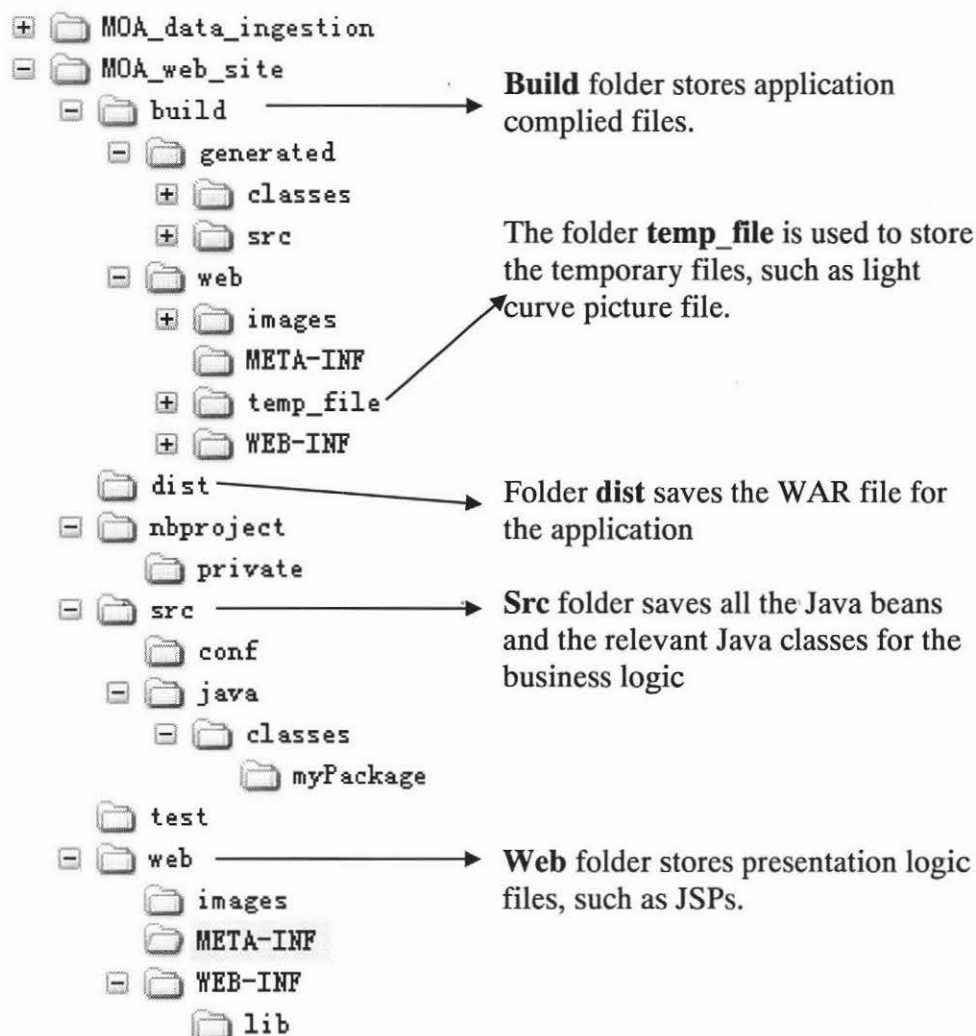


Figure 27: Physical files structure for saving the web application

6.0 Project Performance

6.1 Database performance analysis

The database performance was tested through running one particular query but the relevant tables with different data size through the application. Normally, with the growth of the data size among tables, the process time becomes slower. In order to see how suitable index can improve database performance, the two tests were taken. The first test had been done when there was no any index among tables. In contrast, the second test had been done after the index had been created on table *starIntensity*. From the test results, we can see the suitable index can enhance operation performance.

The testing query:

```
select * from view_star where index=1 and field='ngb16' and CCD=1 and colour='red' and cameraid=5;
```

The index: *Create index starid on starIntensity (indx);*

Table 7: The relevant table size and processing time for the query

TABLE		Process Time(ms)	
<i>Star</i> (Number of data sets in table)	<i>StarIntensity</i> (Number of rows in the table)	NO INDEX	Defined INDEX
112	40613	94	92
2344	886819	30219	9052
8086	3106562	36563	10203
9868	3868935	38813	12828
12969	5234201	58344	26875
14464	5920487	69219	17875
16325	6785041	77609	23109
18381	7792231	97391	28281
23494	10507923	112406	32281
28210	11962950	130016	37422
30276	13307268	162547	48265
33877	14818738	220031	50000
39346	17839861	252087	52350

With the number of data size growth, the performance for querying data can be seen in Figure 28 (Testing environment: P4 1.6, 256MB, windows with PostgreSQL).

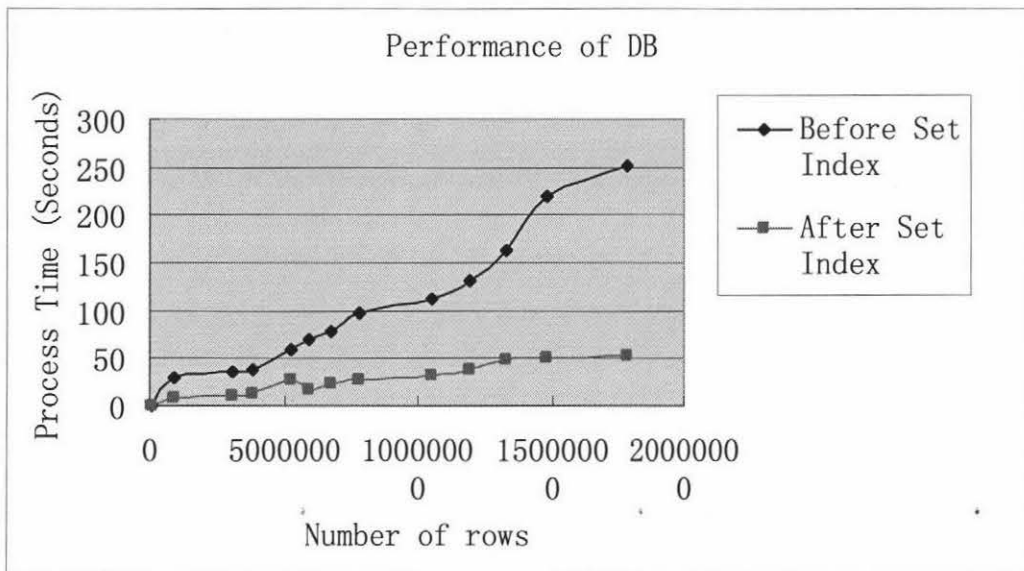


Figure 28: The performance of the database

6.2 Database supporting scientific data mining

The main duty of the database is to support astronomy data mining. David (2001) stated “Data mining is the analysis of observational data sets to find unsuspected relationships and to summarize the data in novel ways that are both understandable and useful to the data owner”. MOA members tried to use obtained data to discover new astronomy knowledge. The process by computer programs to automatically extract useful information from a database is called “knowledge discovery”.

One important example for data mining is to study star light curves. As we know, astronomical objects are far away from us, so the best way for studying astronomical objects is to research star emitted light. Through research on this light, scientists can understand the objects. A light curve is a graph, which shows the brightness of an object over a period of time. NASA members (2006) stated studying of the astronomical objects, which change their brightness over time, such as novae, supernovas and variable stars, the light curve is a simple but valuable tool to scientists. Through studying light curves, astronomers can calculate star light curve period. Then astronomers can schedule the observation time according to star’s period. In addition, the computer program may get the statistic information about the light curves and

then the program can automatically clarify stars according to the statistic information. Therefore, light curve properties for each star are import information for the research. The database created a table called **star** to save star's properties. Each row in the table represents a particular star with properties. The properties include star's identifier, the period of the star light curve and the classification of the star. If stars new properties, such as the mean value of the intensity value, are discovered in the future, they can be easily appended in the table as a new attribute. This information can help MOA to filter data easily. Through the filtered data, MOA can do statistic on astronomical object properties. Furthermore, MOA can study the data mining knowledge. The data mining knowledge can be saved in the Java bean in the application. Under this situation, after the data is taken out from the database, they can apply for data mining knowledge to do analyse.

Now some typical queries for picking data from the tables were shown. These queries can pick data from the database according to MOA research requirements; hence they can support data mining.

This query can get one star's time intensity data (It can be used to produce light curve of the star) within a particular time.

For example: Retrieve the light curve in field is 'ngb1', CCD is 1, index is 3, cameraid is 5, red filter and intensity value between time<2452026 and time>2451640.

```
select * from view_star where field='ngb1' and ccd=1 and colour='red' and
cameraid=5 and index=3 and time<2452026 and time>2451640;
```

This SQL query can filter image data according to its metadata.

For example: The exposure identifier is: field is 'ngb16', Run is 'B1756', CCD is 1 and cameraid is 5

```
select
e.field,e.ccd,e.cameraid,e.colour,e.run,e.jdstart,e.jdend,p.shiftx,p.shifty,p.a,p.b,p.c,
p.d from exposure as e left join pointing as p on e.exposureid=p.exposureid where
e.run='B1756' and e.field='ngb16' and ccd=1 and cameraid=5;
```

The query for filtering stars according to light curve period.

For example: Getting all stars which locate at Field 'ngb1' and star's light curves period<=100 days and period>5day. The results are order by star's index.

```
select * from star where field='ngb1' and period<100 and period >5 order by  
index asc;
```

Find a star's according to its astronomical coordinate system value.

For example: the star position value is RA is 17:58:28.881 and DEC is -29:41:49.59

```
select * from star where ra='17:58:28.881' and dec='-29:41:49.59';
```

This operation can get one particular imaging data's measurement data.

For example: The exposure run is 'B1019' and field is 'ngb11' and list all stars' measurement data.

```
select * from view_star where field='ngb1' and run='B1019';
```

The operation is used for checking how many stars in a particular field.

For example: Field is 1, CCD is 1 and cameraid is 5;

```
select count(*) from star where field='ngb1' and ccd=1 and cameraid=5;
```

6.3 Enterprise application performance


The application was designed based on MVC. It is able to handle many clients' requests simultaneously and give the reply within a short time. This interaction process is illustrated by the screenshots and it is shown by the follows Figures.

The screenshot shows the MOA portal page. At the top, there is a header bar with the text "MOA--Microlensing Observations in Astrophysics". Below this, on the left, is a vertical navigation menu with links: Home, Gravitational, Publication, MOA Telescope, Captured data, About MOA, Discussion Board, and Logout. In the center, there is the Massey University logo and a "Welcome" message. To the right, there is a "Log in" section with input fields for "Student ID:" and "Password:", and buttons for "Sign In" and "Clear". Below the login section, there is a link "Register as New User" and a link "Forgot Your Password? Click here". At the bottom left, there is a copyright notice: "Copyright © 2006 MasseyUniversity".

Figure 29: Portal page for the web site

The screenshot shows the MOA new client registration page. At the top, there is a header bar with the text "MOA--Microlensing Observations in Astrophysics". Below this, on the left, is a vertical navigation menu with links: Home, Gravitational, Publication, MOA Telescope, Captured data, About MOA, Discussion Board, and Logout. In the center, there is the Massey University logo and a "Welcome" message. To the right, there is a "New User" section with a form titled "Please input your detail". The form has input fields for "StudentID:", "First Name:", "Last Name:", and "Email:", and buttons for "Register" and "Clear". Below the form, there is a link "This is the register page". At the bottom left, there is a copyright notice: "Copyright © 2006 MasseyUniversity".

Figure 30: The page for new client registration


MOA--Microlensing Observations in Astrophysics

[Home](#)
[Gravitational Publications](#)
[MOA Telescope](#)
[Captured data](#)
[About MOA](#)
[Discussion Board](#)
[Logout](#)


Copyright © 2006 Massey University

Please input seeking conditions for fetching MOA astronomy data:

Field (such as 16)	<input type="text" value="1"/>	End time:	<input type="text"/>
Ccd (such as 1)	<input type="text" value="1"/>	RA:	<input type="text" value="17:58:30.177"/>
Index (such as 5)	<input type="text" value="1"/>	DEC:	<input type="text" value="-29:38:33.62"/>
Start time:	<input type="text"/>	Distance:	<input type="text"/>
CameraID:	<input type="text" value="5"/>	<input type="button" value="Submit"/>	
Please choose wanted result type		<input type="button" value="Get data by table"/>	

See the Star Light Curve by picture [Please click here](#)

Figure 31: The page for fetching astronomy data under general operation.
 (User can get star time intensity value based on star's ID, position and time)


MOA--Microlensing Observations in Astrophysics

[Home](#)
[Gravitational Publications](#)
[MOA Telescope](#)
[Captured data](#)
[About MOA](#)
[Discussion Board](#)
[Logout](#)

Copyright © 2006 Massey University

Please input the query command by yourself.

There is a table view called viewlight, you only can use 'select' command

Figure 32: The page for fetching astronomy data under professional operation
 (User can filter data dependent on their own SQL query)

MOA--Microlensing Observations in Astrophysics									
	CAMERA	ID	FIELD	COLOUR	CCD	INDEX	RUN	TIME	POS X POS Y FLUX ERROR
Home	5		ngb16	red	1	1	B3200	2452741.13217	207.189 546.543-2203.332108.65
Gravitational	5		ngb16	red	1	1	B3206	2452743.152992	207.189 546.5431804.01 1642.54
Publications	5		ngb16	red	1	1	B3211	2452749.05636	207.189 546.5439807.19 1908.46
MOA Telescope	5		ngb16	red	1	1	B3215	2452750.039404	207.189 546.5439240.05 3207.52
Captured data	5		ngb16	red	1	1	B3259	2452759.13923	207.189 546.54317393.7 1066.51
About MOA	5		ngb16	red	1	1	B3263	2452762.245978	207.189 546.54321480.1 1527.2
Discussion Board	5		ngb16	red	1	1	B3268	2452765.05952	207.189 546.54320337.2 867.394
Logout	5		ngb16	red	1	1	B3275	2452767.200868	207.189 546.54317134.1 1469.14
	5		ngb16	red	1	1	B3351	2452775.048906	207.189 546.54310194.5 2151.05
	5		ngb16	red	1	1	B3360	2452776.1052895	207.189 546.5439760.11 952.473
	5		ngb16	red	1	1	B3368	2452777.0293575	207.189 546.5439463.89 7797.34
	5		ngb16	red	1	1	B3381	2452784.197413	207.189 546.543652.981 2192.2
	5		ngb16	red	1	1	B3390	2452790.1665165	207.189 546.5433025.23 2205.32
	5		ngb16	red	1	1	B3408	2452794.1755675	207.189 546.54312722. 1798
	5		ngb16	red	1	1	B3412	2452795.1119965	207.189 546.54313287.6 800.291
	5		ngb16	red	1	1	B3485	2452801.089149	207.189 546.54324914.7 1615.58
	5		ngb16	red	1	1	B3517	2452802.065486	207.189 546.54326363.9 1218.77
	5		ngb16	red	1	1	B3530	2452803.0534835	207.189 546.54328320.1 1982.4
	5		ngb16	red	1	1	B3602	2452806.968721	207.189 546.54327519. 1424.46
	5		ngb16	red	1	1	B3604	2452807.123698	207.189 546.54326190.6 2185.49
	5		ngb16	red	1	1	B3606	2452807.281418	207.189 546.54326530. 1485.2
	5		ngb16	red	1	1	B3697	2452826.933177	207.189 546.543-5024.731089.1
	5		ngb16	red	1	1	B3698	2452827.004693	207.189 546.543-6504.511564.17
	5		ngb16	red	1	1	B3702	2452828.043895	207.189 546.543-2510.711097.03
	5		ngb16	red	1	1	B3712	2452830.040602	207.189 546.543-4051.271933.47

Copyright © 2006 Massey University

[next page](#) [Previous page](#)

Figure 33: The page showing the fetching data by text table

MOA--Microlensing Observations in Astrophysics									
Current logged In as: Paul Mied 87897536									
Home									
Gravitational									
Publications									
MOA Telescope									
Captured data									
About MOA									
Discussion									
Logout									
	RUN	FIELD	COLOUR	CCD	JDSTART	JDEND			
	B1466	ngb16	red	1	2452134.12809	2452134.130243			
	B1467	ngb16	red	1	2452134.13485	2452134.137002			
	B1468	ngb16	red	1	2452134.141597	2452134.143762			
	B1469	ngb16	red	1	2452134.1511	2452134.153264			
	B1470	ngb16	red	1	2452134.157859	2452134.160023			
	B1504	ngb16	red	1	2452137.107118	2452137.109271			
	B1505	ngb16	red	1	2452137.110625	2452137.112789			
	B1506	ngb16	red	1	2452137.11537	2452137.117535			
	B1507	ngb16	red	1	2452137.118889	2452137.121053			
	B1508	ngb16	red	1	2452137.123461	2452137.125613			
	B1509	ngb16	red	1	2452137.851296	2452137.853449			
	B1510	ngb16	red	1	2452137.921968	2452137.924132			
	B1511	ngb16	red	1	2452137.984931	2452137.987095			
	B1512	ngb16	red	1	2452138.058241	2452138.060394			
	B1513	ngb16	red	1	2452138.121632	2452138.123796			
	B1514	ngb16	red	1	2452139.927211	2452139.929375			
	B1515	ngb16	red	1	2452139.994479	2452139.996632			
	B1516	ngb16	red	1	2452140.062176	2452140.06434			
	B1517	ngb16	red	1	2452140.12912	2452140.131285			

Copyright © 2006 Massey University

Figure 34: The table showing the imaging metadata information

MOA--Microlensing Observations in Astrophysics

[Home](#)
[Gravitational](#)
[Publications](#)
[MOA Telescope](#)
[Captured data](#)
[About MOA](#)
[Discussion](#)
[Logout](#)

Please input details for retrieving light curve:

Field: <input type="text" value="1"/> * (Such as: 16)	RA: <input type="text" value="17:58:30.177"/>	DEC: <input type="text" value="-29:38:33.62"/>
CCD: <input type="text" value="1"/> * (Such as: 1)	Time specify: <input type="radio"/> No <input type="radio"/> Yes	Grid: <input type="radio"/> No <input type="radio"/> Yes
Index: <input type="text" value="5"/> * (Such as: 1)	Start time: <input type="text" value="2450000"/>	Error Bar: <input type="radio"/> Yes <input type="radio"/> No
Filter: <input type="text" value="red"/> *	End time: <input type="text" value="2550000"/>	Tolerant error value: <input type="text" value="50000"/>
CameraID: <input type="text" value="5"/> *	<input type="button" value="Draw"/> <input type="button" value="Reset"/>	

Copyright © 2006
 Massey University

Figure 35: Screen shot of inputting parameters for the light curve
 (User can get light curves by pointing out star's OID or location, the period and the display picture format.)

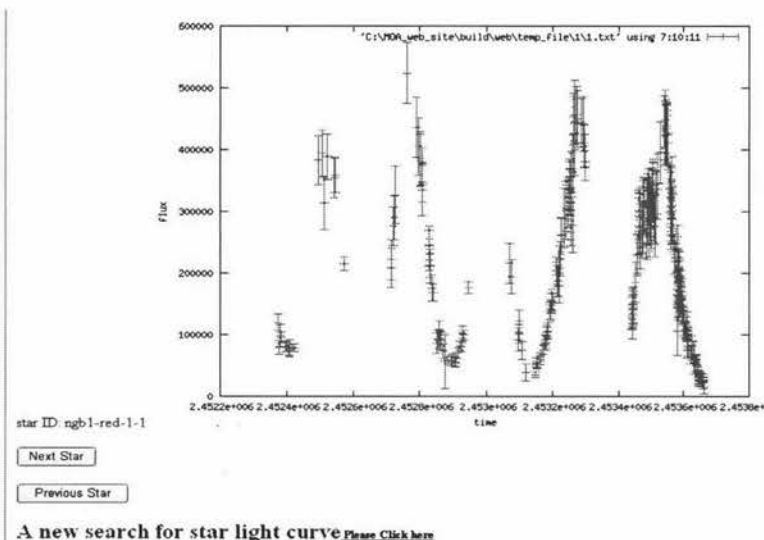
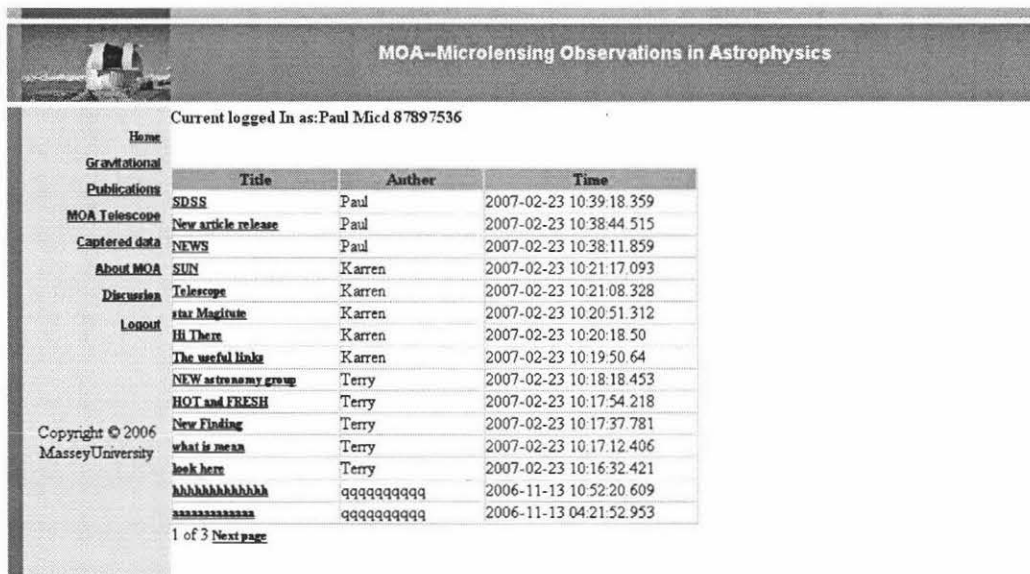


Figure 36: Screen shot of a star light curve



MOA--Microlensing Observations in Astrophysics

Current logged In as: Paul Micd 87897536

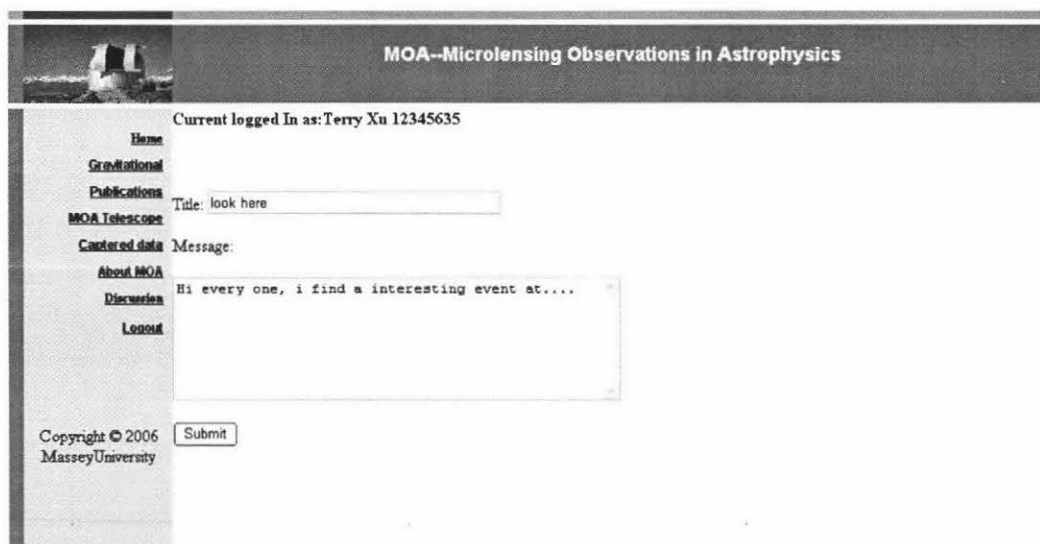
Home
Gravitational
Publications
MOA Telescope
Captured data
About MOA
Discussion
Logout

Title	Auther	Time
SDSS	Paul	2007-02-23 10:39:18.359
New article release	Paul	2007-02-23 10:38:44.515
NEWS	Paul	2007-02-23 10:38:11.859
SUN	Karren	2007-02-23 10:21:17.093
Telescope	Karren	2007-02-23 10:21:08.328
via Magnitude	Karren	2007-02-23 10:20:51.312
Hi There	Karren	2007-02-23 10:20:18.50
The useful links	Karren	2007-02-23 10:19:50.64
NEW astronomy group	Terry	2007-02-23 10:18:18.453
HOT and FRESH	Terry	2007-02-23 10:17:54.218
New Finding	Terry	2007-02-23 10:17:37.781
what is mean	Terry	2007-02-23 10:17:12.406
look here	Terry	2007-02-23 10:16:32.421
hhhhhhhhhhhh	qqqqqqqqqq	2006-11-13 10:52:20.609
aaaaaaaaaaaa	qqqqqqqqqq	2006-11-13 04:21:52.953

Copyright © 2006
Massey University

1 of 3 [Next page](#)

Figure 37: Screen shot of the discussion board



MOA--Microlensing Observations in Astrophysics

Current logged In as: Terry Xu 12345635

Home
Gravitational
Publications
MOA Telescope
Captured data
About MOA
Discussion
Logout

Title:

Message:

Copyright © 2006
Massey University

Figure 38: Screen shot of posting a subject to the discussion board

7.0 Conclusion

Modern astronomy is gathering more information now than at any other period in the past. The ability to analyze and to draw conclusions from this large volume of data will be an ever-constant challenge for astronomers. A good database system based on modern computer technologies can help astronomers to discover the astronomical phenomenon. In this project, a database and enterprise application for the MOA group was created. After clarifying the inter-relationships of the obtained data, this database can save astronomy data and support for the further analysis activities-data mining. The enterprise application was designed based on MVC pattern. This is able to publish MOA's findings, share astronomy data and provide a place for people's discussion on the Internet, but also supports scientific data mining based on MOA obtained data. Due to security consideration, such as the Internet hacker, an isolated application was created for MOA to manage the database such as update and modify the database. The database was tested on a 7 GB subset of the achieved MOA data set, which are over 17 million records. Through testing, we can see the application can quickly retrieve the data and take strengths of the PostgreSQL database system.

Through the whole development of the project, various numbers of technologies have been studied and practised. The main three technological components of this project are Java software programming, Java Server Page programming, and PostgreSQL database language. The application was written by Java. Therefore, it can be used in any platform. The new database can afford trillions of the data with variety of operations. And the application was created based on MVC design pattern; it separates the web application into three tiers including Model, View and Controller. Therefore, the web site can be extended and maintained easily.

8.0 Future Work

At the point, this prototype was designed for MOA to save their accumulated data under the current situation. With development of research requirements, part of the prototype can be improved in the future.

1. In observational data volumes, the data can be saved in a distributed database with distributed database management system to control.
2. If the obtained data type or data relationship becomes more complex, the data can be managed by object relational database manage system. The dimensional model under data warehouse concept can use to model the data, the data will be easily summarized using OLAP query.
3. The prototype has saved imaging metadata in the database. The searching imaging data can be operated by SQL using metadata table, which contains information of FITS (imaging file) header parameters. But after the prototype retrieves the imaging file name from the table, it needs a suitable application to get physical imaging file as fast as possible. How these imaging files saving on the storage (The physical structure) and how to quickly access them are expected by the future study.
4. At the moment, the database is backuped manually. In the future, this can apply for PostgreSQL function “on-line backup and point-in-time recovery” to backup and recovery the data more smartly.
5. The application has three tiers at the moment. It can be separated into more tiers if it is necessary. And it can use Enterprise Java bean to control the data flow and business logic between the database and the application.
6. At the moment, the application doesn't include any astronomical data mining knowledge. But it can be added in the future. The basic idea is to save data mining knowledge in the Java bean. And then the Java bean takes data out from the database and applies for data mining knowledge, such as an algorithm to calculate light curve period or statistic star properties. Lastly, the Java bean returns the process results to the clients.
7. The prototype was tested on a subset of the MOA data. If the entire dataset is loaded in, the performance should be measured again. It is a limitation of this project.

References

- American Association of Variable Star Observers. (2006). *Variable stars*. Retrieved on Feb 15, 2007 from: <http://www.aavso.org/>
- Astronomical Institute (SAI). (2007). *SAI Database*. Retrieved on Jan 2, 2007 from <http://www.sai.msu.su/database.html>
- Baruffolo, A., & Benacchio, L. (1998). Object-relational DBMSs for Large Astronomical Catalogue Management. Astronomical Observatory of Padova, Italy. *Astronomical Data Analysis Software and systems VII. ASP Conference Series, Vol. 145*.
- Chilingarian, I., Bartunov, O., Richter, J., & Sigaev, T. (2004). PostgreSQL, the Suitable DBMS Solution for Astronomy and Astrophysics. *Astronomical Data Analysis Software and System XIII, Vol. 414*.
- Chris, S. et al. (2002). Real-time time-variability Analysis of GB to TB Datasets: Experience from SuperMacho and Supernova project at NOAO/CTIO. *Survey and Other telescope technologies and Discoveries, Vol. 4836*.
- Dennis, R., & Daniel, D. (1994). The archives of Canadian Astronomy Data Centre. Canadian Astronomy Data Centre. *Dominion Astrophysical Observatory*. Retrieved on Nov 1, 2006 from http://cadwww.dao.nrc.ca/ADASS/adass_proc/adass3/papers/crabtreed/crabtreed.html
- Dustin, M. (2001). *JSP best practices*. Retrieved on Nov 1, 2006 from <http://www.javaWorld.com>
- George A. (1997). *Exploration of the Universe*. Retrieved on Nov 1, 2006 from http://imagine.gsfc.nasa.gov/docs/science/how_11/images.html
- Gnuplot homepage (2006). Retrieved on Jan 8, 2007 from <http://www.gnuplot.info/>
- Haque, I., & O'Connor, B. (2002). *J2ME Enterprise Development*. M&T Books New York, NY 10022.
- Huggins, J. (2007). *The Astronomy Net*. Retrieved on Jan 8, 2007 from: <http://www.astronomy.net/about/history.html>
- Hiriart, R., & Smith, C. (2003). *The SuperMacho+SuperNova Survey Database Design: Supporting Time Domain Analysis of GB to TB Astronomical Datasets. Astronomical Data Analysis Software and Systems XII, Vol. 295*.
- John, R. (1996). Starbase: A User Centred Database for Astronomy. *Astronomical Data Analysis Software and System V, Vol. 101*.
- Matthew, W. (2002). *Digital Dig- Data Mining in Astronomy*. Retrieved on Sep 30, 2006 from <http://www.astrosociety.org/pubs/ezine/datamining.html>

Morrison, J., & Morrison, M. (2003). *Guide to ORACLE9i*. Senior Vice President, Publisher.

Nasa/Ipac Extra galactic Database. (2007). Retrieved on Feb 1, 2007 from <http://nedwww.ipac.caltech.edu/>

Kifer, M., Bernstein, A., & Lewis, P.(2006). *Database Systems: an application-oriented approach*. (2nd ed.). Pearson Education, INC.
Kmiec, M. (2002). *Does J2EE live up to expectations?* Retrieved on Nov 1, 2006 from <http://news.zdnet.co.uk/software/0,10000000121,2121919,00.htm>

MOA. (2006). *The MOA Project Web site (Auckland University)*. Retrieved on Aug 10, 2006 from: <http://www.physics.auckland.ac.nz/moa/index.html>

Oracle. (2006). Retrieved on Oct 20, 2006 from <http://www.oracle.com/index.html>

Paul, D.(2005). *Mysql Query Optimization*. Retrieved on Nov 8, 2006 from <http://www.informit.com/articles/article.asp?p=377652&seqNum=4&rl=1>

PostgreSQL Global Development Group.(2006). *PostgreSQL 8.1.0 Documentation*. Retrieved on Jan 20, 2007 from: <http://www.postgresql.org/>

Roman, S. (1997). *Access Database Design & Programming*. O'Reilly & Associates, Inc, CA 95472.

Sloan Digital Sky Survey/ SkyServer(SDSS). (2006). Retrieved on Nov 1, 2006 from <http://cas.sdss.org/dr4/en/sdss/>

Sun.(2006). *Sun Developer Net word (SDN)*. Retrieved on Jan 20, 2007 from <http://java.sun.com/>

The Filing System. (2006). Retrieved on July 5, 2006 from http://www.flirble.org/chrisy/vmm386/user-man/user-man_4.html

The Royal Observatory Edinburgh.(2005). Retrieved on July 25, 2006 from: <http://www.roe.ac.uk/>

Thomas, C., & Carolyn, B. (2005). *Database Systems. A practical approach to design, implementation and management*. (4th ed.). Pearson Education Limited.

Thomas, P. (1999). *HTML: The Complete Reference*. (2nd ed.) Brandon A. Nordin.

Wenger, M., Kinnar, F., & Jocqueau, R. (2002). *SIMBAD as a Test Bed for two Object Oriented Database Management Systems: Objectivity/DB and O2*. *Astronomical Data Analysis Software and Systems IX*, Vol. 247

APPENDIX A -- SQL for creating MOA database

```
create database moadata;  
\c moadata;
```

```
create table camera (cameraid int not null primary key, name char(10),step int, naxis1  
int, naxis2 int, numberccd int, telescope char(5));
```

```
create table calibration (cameraid int not null, field char(6) not null,ccd int not null, ra  
char(15),dec char(15),n1 float,n2 float,n3 float,n4 float,n5 float,n6 float,n7 float,n8  
float,primary key (cameraid,field,ccd), foreign key (cameraid) references camera  
(cameraid) on update cascade on delete cascade);
```

```
create table exposure (exposureid serial primary key, run char(6), field char(6),  
colour char(5), ccd smallint not null,jdstart float, jdend float, cameraid int not null,  
foreign key (field,ccd,cameraid) references calibration (field,ccd,cameraid) on update  
cascade on delete cascade,unique(run,field,colour,ccd,cameraid));
```

```
create table pointing (exposureid int not null references exposure primary key on  
update cascade on delete cascade,shiftx float,shifty float,a float, b float,c float,d  
float,cameraid int);
```

```
create table star (starid serial primary key,field char(6),ccd int,index int,colour char(5)  
default 'red',cameraid int,period float default 0, classification char (12),ra char(15),dec  
char(15), pos point, posshift point,unique(field,ccd,index,colour,cameraid));
```

```
create table starintensity (staristid serial primary key, starid int,time float, run char(6),  
flux float, error float, n1 float, n2 float, n3 float, n4 float, n5 float,exposureid int,  
foreign key (starid) references star (starid), foreign key (exposureid) references  
exposure (exposureid));
```

```
-----  
Create index locationid on star using btree (RA, DEC);  
Create index starid on star using btree (field,ccd,colour,cameraid,index);  
Create index timeid on starintensity using btree (time);  
Create index star_c_id on star using hash (classification);  
Create index star_p_id on star using btree (period);
```

```
create view view_star as select  
s.field,s.ccd,s.colour,s.cameraid,s.period,s.classification,s.index,s.ra,s.dec,i.flux,i.error  
,i.run,i.time, i.n1,i.n2,i.n3,i.n4,i.n5 from star as s, starintensity as i where  
s.starid=i.starid;
```

```
-----  
create database member;  
\c member  
create table student (st_studid char(8) NOT NULL default "", st_fname char(30)  
default NULL, st_lname char(30) default NULL, st_email char(40) default NULL,  
st_password int default NULL, PRIMARY KEY (st_studid));
```

```
create table discussion (id serial primary key, title char(30) NOT NULL default
",author char(20) NOT NULL default ",time timestamp, content text);
```

```
create table disdetails (detailsid serial primary key, id int not null, author char(30)
NOT NULL default ", time timestamp, content text ,foreign key (id) references
discussion (id) on update cascade on delete cascade);
```

```
insert into student values (12345678,'Tester','MOA','test@hotmail.com',1234); --set
up a normal role for testing
```

```
insert into student values (11111111,'Administer','MOA','admin@hotmail.com',1111);
--set up Administer role
```