UNIVERSITY OF HERTFORDSHIRE

Faculty of Science, Technology and the Creative Arts

Modular BSc Honours in Computer Science

6COM0282 – Computer Science Project

Final Report

April 2012

School Timetabling Management System

V.R. Carrell

Supervised by: Neil Davey

Abstract

This report looks at the building of a database and the creation of a user interface to store data for a school which is capable of dealing with the timetabling of students and teachers but is primarily able to locate individuals in the event of an emergency.

To achieve this aim accurate information needed to be gathered which was needed to produce a working system that would be capable of with solving this problem. Research was also needed to discover how to integrate a timetabling system with the information of individuals and how these details could be shown in a database. This would then be integrated with a usable interface to allow the users of the system to access the details efficiently.

From the research and requirements analysis of the timetabling system, the appropriate use cases were identified and these were achieved through a functional database which was built within the time constraints of the project. A working java user interface was unable to be constructed to work in conjunction with the database.

Contents

Table of Figures

Index of Tables

Glossary

Before reading this report there are some terms that need to be understood in relation to my project:

Attributes	A named column of a relation
Data	Any fact that needs to be stored
Domain	A set of allowable values for one or more attributes
DBMS	Database Management System
Foreign key	An attribute or set of attributes, within one relation that matches the primary key of the same table or another table.
Null	Represents a value for an attribute that is currently unknown or is not applicable for this tuple.
Primary Key	the candidate key that is selected to identify rows uniquely within the relation
Relation	A relation is a table with columns and rows.
Relational database	A collection of normalised relations with distinct relation names.
Relation Schema	A named relation defined by a set of attributes and domain name pairs.
Relational database schema	A set of relation schemas, each with a distinct name.
SQL	Structured Query Language

Introduction

During my industrial placement I worked at a secondary school, the school needed to deal with the timetable and monitoring of students which was an extremely important part of the daily activities of some of the staff members. There was a separate timetabling system from a student information system and if these details were integrated the schools timetable management could be produced more efficiently.   The system didn't currently allow the search of a student to find out where they are located in the school, they need to go on a student's form name and look at printed timetables to find this information out. Noticing these issues I was really interested in how I could create this system, as I believed the system the school used could be more advanced.   I wanted to create a more sophisticated system which could locate where a particular student is at a certain time or even find a teacher who could either be teaching or in a free period. To also have these details in the case of an emergency would be extremely useful as it would allow someone to find the location of someone situated in different part of the school by just searching for their name in a system.

From my placement I started thinking of ways how this software could actually be created, I also became interested in what kinds of information would need to be captured to make it work. When returning to university for my final year I was reminded that for computer science students there is no way of getting a personalised timetable and that for some lessons e.g. practical lessons and tutorials there is no details kept on a system to inform lecturers which students should be attending their particular classes. This made me more interested in how I could create a system that would be capable of producing individual timetables.

1.1 Aims

The aims of my project were:

- To create a student/ teacher timetabling system.
- To design and implement a database system to assist in the timetabling of classes in an academic institution/ school.
- To create a database that will deal with the scheduling of lecturers/ teachers to timetables.
- To create an information system, where in the case of emergency someone working for the school can find out where individual students are located.
- To create a Student Information System with the ability to search the students name and retrieve their individual timetable, and where they can be located.
- To create a system that is capable of recognising clashes
- To deal with the distances of rooms and not allocate rooms too far away for a teacher or student to be able to get to in a short amount of time.

1.2 Objectives

To meet the aims of my project, I devised a list of objectives to work towards which are listed below.

Background research:

- Research the required functionality needed in completing this system.
- Research how I will create a system that will fulfil the aims of my project.
- To research database software that is the most appropriate to use.

Requirements Analysis:

- Identifying use cases and actors
- Identify the functional requirements of the system
- Identify the non-functional requirements of the system

Design:

- Prototype of what the application will look like.
- ER diagram modelling the relationships of the database.
- Details of how the system should work.

Implementation:

- Create a database, which has data stored in it.
- Deal with data integrity.
- How I can overcome the difficulties in dealing with name clashes, different subject areas.
- An application which allows the user to carry out the requirements of the system.
- A system that can deal with clashes of lessons.

Testing:

- Testing of the database and checking for referential integrity and other issues there may be.
- Carry out queries on the database to check the requirements of the project have been met
- Testing of the application and what it showed, if any problems were noticed and how they were resolved.
- Create advanced queries that show how my system works.
- Testing of the user interface and checking it meets the requirements.

1.3 Report Structure

The report is structured as follows:

- Chapter 2 discusses the research done during the project and the decisions that I made.
- Chapter 3 discusses the requirements analysis of the project.
- Chapter 4 discusses the design, implementation, testing and evaluation of my database.
- Chapter 5 discusses the design, implementation, testing and evaluation of the user interface.
- Chapter 6 provides the technical and personal conclusion of my project.
- Appendix A contains the Gantt chart which was designed prior to the project.
- Appendix B contains the use case descriptions designed in the requirements analysis stage of the project.
- Appendix C contains the MySQL code for creation of the database tables.
- Appendix D contains the Insert into methods for MySQL
- Appendix E contains the Alter table methods for altering the database tables.
- Appendix F contains the queries used to check the requirements of the database were met.
- Appendix G contains the test data that was inserted into the database.
- Appendix H contains the java code for the java application.

2. Research and Background

2.1 Gantt Chart

The planning of my project was done by creating a Gantt chart (Appendix A)
The main processes of my project were an analysis, implementation and testing.
The first stage, Analysis included my research and the planning of my system and some time learning SQL.
The second stage, Implementation involved building my database, creating the user interface and then linking the database to the program.
The third stage of my plan was the testing of the program.
The fourth stage of the plan was the writing of the report.

2.2 Research

Before starting my project I didn't have much experience with databases, one of the reasons I was so interested in this project was because I wanted to be able to confidently design and create a database before leaving university.

I decided on making a java application for my user interface as I had previous experience of java from course modules I have taken. I chose this programming language as it is one of the most well-known programming languages and there are so many available possibilities that can be done with it. Another advantage being that there is also a driver that will allow me to connect the java to MySQL; which is called a JDBC driver.

I chose to create a database as it is the best way of storing data and allows advanced queries. The database technology I decided to use was SQL which is a programming language that deals with the managing of data in a relational database management system. The reason I chose SQL was that there were many places to access online resources regarding the building of database systems; it is also the international standard language for manipulating relational databases (Russell, 2012). After researching different types of databases interfaces my main choices were Oracle or MYSQL. After testing out the two types, although both very useful I decided on MYSQL as I preferred the use of the command line editor and found I didn't have to spend too much time learning how to use a new interface.

Initially when starting this project I had minimal knowledge of databases and how they worked. I chose MySQL due to the amount of resources available to help people learn how to use SQL. My research involved teaching myself about databases and how I could approach my project the best way. My time was spent following tutorials that had been set up online and also reading around the subject. Once I had completed a number of tutorials I was able to put my project into context and start thinking about how I was going to approach the problem.

From my research into how to create databases I needed to understand what a database actually did and what it was capable of before I could make my project, this meant that I would fully understand most concepts needed to do with databases before I started creating one, by following tutorials I managed to learn the basic concepts of how to create databases and how to administer them.

2.2.1 An introduction to databases

A database is a repository of data which can be used simultaneously by many departments and users. A database management system (DBMS) is a software system that allows users to define, create, maintain and control access to the database (Basics of Computer, 2012). There are many advantages to using DBMS such as data consistency, enforcement of standards, good chances to analyse data and many more. The disadvantages of DBMS can be the complexity of designing them, their size and they can be expensive to build (Connolly, 2002). For my project a database is the best option, this is because for me to be able to achieve my project aims, I need to be able to store large amounts of data and link this data together to enable me to get the correct results. In a database the data that is captured is held in tables, which are also called entities. An entity is a business object, it can either represent a tangible product e.g. an item or an intangible one e.g. an event (Rational Software Corporation, 2002). In my database every entity must have a different name and each entity will have attributes that are able define it. An entity will become a table in my database and an attribute will become a field in that table. There are two types of attributes; a non-key attribute describes the entity and a key attribute which identifies the entities (Rhodes, 2011).

The Data Manipulation Language (DML) and Data Definition Language (DDL) exist to manipulate and create databases. DDL is used to define the database and create entities and input data into databases (Techopedia, 2010). DML syntax are functions that are in the form of a verb (E.g. INSERT INTO …. VALUES …., SELECT FROM). These functions are necessary when creating and developing databases as they are the programming language to insert and manipulate data in the database.

During my research I encountered issues with many-to-many relationships in my relational database management system RDBMS, most do not allow these types of relationships (Database Dev, 2003). These will be dealt with by using a link table (also known as a junction table), by using a link table resolves the issue of many-to-many relationship by placing a table in the middle of the two entities and using the primary keys in the surrounding table as foreign keys in the link table.

There are five types of integrity constraints to consider ensuring that data entered into a database is precise, valid and reliable (Connolly, 2002).
Listed are the different constraints that I may come across when designing my database (Oracle® Database , 2008).

- Required data is where some attributes must hold a value, so nulls cannot always be accepted in a database, this can also be called a 'not null constraint'.
- Attribute domain constraints defines the data type and length restricting the user to only put in defined values.
- Entity integrity (primary key constraint) defines a primary key to each entity to ensure that when inserting data into a table, no duplicate primary key is created, it also means a primary key cannot hold null values.
- Referential integrity (foreign key constraint) ensures that any foreign key declared in a table can only contain values created from a primary key in the parent table.
- Enterprise integrity (check constraint) is used to make a value satisfy a specific condition, this can be used when setting business rules e.g. a room cannot have more than thirty people.

Database design is made up of three phases: conceptual, logical and physical design (Microsoft, 2005). I decided to use these to help me design my database as it is one of the most popular tools for designing databases. The conceptual database model highlights the real world activities that the users of the system need to carry out, it will also highlight the actors and stakeholders of the system. The logical database design is about translating the conceptual design of the database into a logical design, this is so that when the physical part of the database is designed this would help me make the correct decisions when I started building my database (Connolly, 2002). At the end of completing the three phases of design I should clearly be able to decide on how the database will be created.

2.2.2 An Introduction to Java

To help me decide how I was going to create my java application, I followed different tutorials online that offered me chances to familiarise myself with the way other people chose to connect their databases to a java application. I chose to use the software Netbeans IDE which is an integrated development environment which enables the development of creating a java application. The reason for choosing this software was because I was familiar with it from one of my course modules and it meant that I didn't have to teach myself to use new software.

The java system that I decided to create was a java application.

Java Database integration:

There are a few ways of integrating databases in java. I decided to use the Java Persistence API. The reason I chose this was because it is meant to be simpler to use than Java hibernate. As I didn't have any experience of linking java with a database I thought this was the best option for me.

A Java JPA API is ideal for the needs of my database as it allows access, persistence and management of data between java objects/ classes and a relational database management system (RDBMS). This is ideal for me to connect my database to enable me to make my application (ObjectDB, 2010).
There are three artefacts that make a JPA compliant program which are an entity class, a persistence.xml file and a class through which will insert, update, or find an entity (Das, 2008).
An entity class is created which represents a table in a database, within the class there are identifiers that tell us that it is connected to a database.

- @Entity signifies that this class is an entity class
- @ID means it is a primary key
- @Column refers to the columns in the tables of a database.

The persistence.xml file XML file placed in the META-INF folder of the created application is used to specify the persistence provider name, entity class names, and properties like the database connection URL, driver, user, and password (Venkatasamy, 2012). To create the entity classes is simple in the Netbeans IDE, one of the reasons I chose to use to use this platform is that it can auto generate code, although I might be able to write the code, it saves a lot of time if it is generated automatically.

The interface:

For the interface of my application, I have chosen to use Java Swing. The reason I chose to use swing is because it makes designing an interface simple, the tool allows an interface to be easily created and designed by just dragging and dropping in anything from buttons to tables, which could then be configured to perform the way I wanted (Oracle, 2012). It uses model-view-controller architecture which is an architectural pattern, the model deals with the business logic, the view deals with the user interface and the controller is responsible for dealing with the users inputs (Deacon, 2009). By following tutorials and watching videos online, I was able to see what possibilities there are in building user interfaces practice how I would be able to create it.

2.3 The School

The following are the rules behind the assembly of my system; these take into account the details of my system. From my placement I was able to gain knowledge of the working environment of the school and how they managed different areas of the school.

My project is based on a secondary school with the year groups of children ranging from years seven to thirteen. Each day has six periods of lessons and in a week there will be a total of thirty lessons. This is in compliance to the way most secondary schools teach their lessons currently in the UK.

Within the schools structure there are staff members teaching from seven different departments overall: English, Humanities, Languages, Science, PE, Performing Arts, and Design Technology. Within the departments are the following subjects: Maths, English, History, French, Spanish, Science, Drama, Music, PE, and Technology.

There is a maximum of thirty students in a class, this is because the school does not allow more than thirty students in a classroom at a time and this is to help students have a healthy working relationship with their teacher. A student must attend all thirty lessons in a week, whilst a teacher can teach less than thirty. The amount of subject lessons a student attends depend on which subject it is e.g. a student will attend more Maths lessons a week than they would Drama lessons.

The details above are needed to consider what information is needed when I create my system, I will discuss this further in my requirements analysis.

3. Requirements Analysis

The following chapter discusses how I developed the requirements analysis for my system

Before starting the design of my project, I needed to collect the factual evidence of what information I needed and was relevant to my project (2002: Begg, Connolly).

Section 2.3 of my report describes how the school operated; this information helped me to complete the requirements analysis of my system.

Requirements analysis is about understanding the user's needs and documenting them before actually designing the product (Maguire, 2002). There are two types of requirements which are functional and non-functional. Functional requirements deal with what the software actually does whereas non-functional requirements deal with how well it does it, e.g. usability, maintainability (Bredemeyer & Malan, 2001).

The requirements analysis of a project is extremely important; if the correct requirements are not captured in the start of the project this could mean that the program will not achieve its aims. It will help me produce the requirements needed for my database and my java application.

3.1 Functional Requirements

From my research (Evans, 2004) I was able to determine how I would find my requirements for my system, I have had experience of carrying out requirements analysis during my course already so I already had some knowledge of the process.
To gather my functional requirements I first determined the actors of the database, these are considered to be the stakeholders of the systems as they are affected by it. I then decided the use cases which define interactions between an actor and the system to achieve an objective. After this I decided to present the use cases in use case diagrams to represent what the system will do and how the actors interact with it. I chose these methods because during my university course it is the form of requirements capture I became most familiar with.
By first determining the actors of the system, I was then able to figure out the use cases of my project and confirm what functions I needed my database and java application to be able to do.

Actors

- **User**: is a person who interacts with the system and can view and add students into the school. They can perform searching functions e.g. to find what lesson a student is in.
- **Administrator**: person who can interact with the system via the server interface and can make changes to it such as add/edit user details and add/remove students and staff members from the database.

Use Cases

User:
- Add student details
- Edit student details
- Add staff member
- Edit staff member
- Search for student/ staff details

- View all student/staff/ lessons/ equipment

Administrator:

- Add user details
- Edit user details
- Remove user
- Remove student/ staff/ rooms/ equipment/ department/ subject/ events
- Add new student/ staff/ rooms/ equipment/ department/ subject/ events

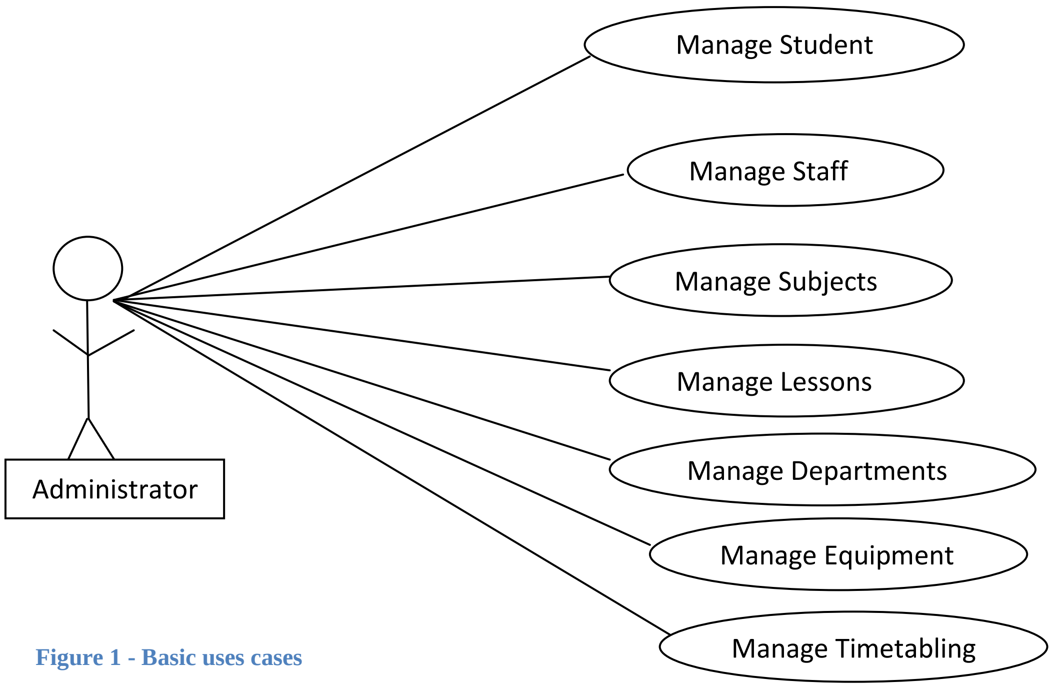


Figure 1 - Basic uses cases

The Use Case Diagram represents the system administrator's basic interaction with the system. This diagram shows that the administrator will fundamentally be involved in every part of the system.

Before discussing the non requirements of the system there were a few things that needed to be made clear.

The quality of data is incredibly important in this project. As it is timetable management software that the times lessons are run are specific and also the system needs be able to deal with clashes so that there aren't any errors (this will be dealt with in the java application). If there are discrepancies between data in my project then it may not work as it should. The risks associated with incomplete data could mean my project doesn't respond to queries the way it should and that there could be further issues that develop from this. As my system will store personal information it will need to be kept up to date under the Data Protection Act (UK, 1998). The person responsible for the updating of information would be decided by the school, but organisations tend to have someone in charge of data protection. This means that I need to make sure the system I create is editable for users to make changes if they need to, so they are able to comply with the Data Protection Act (UK, 1998).

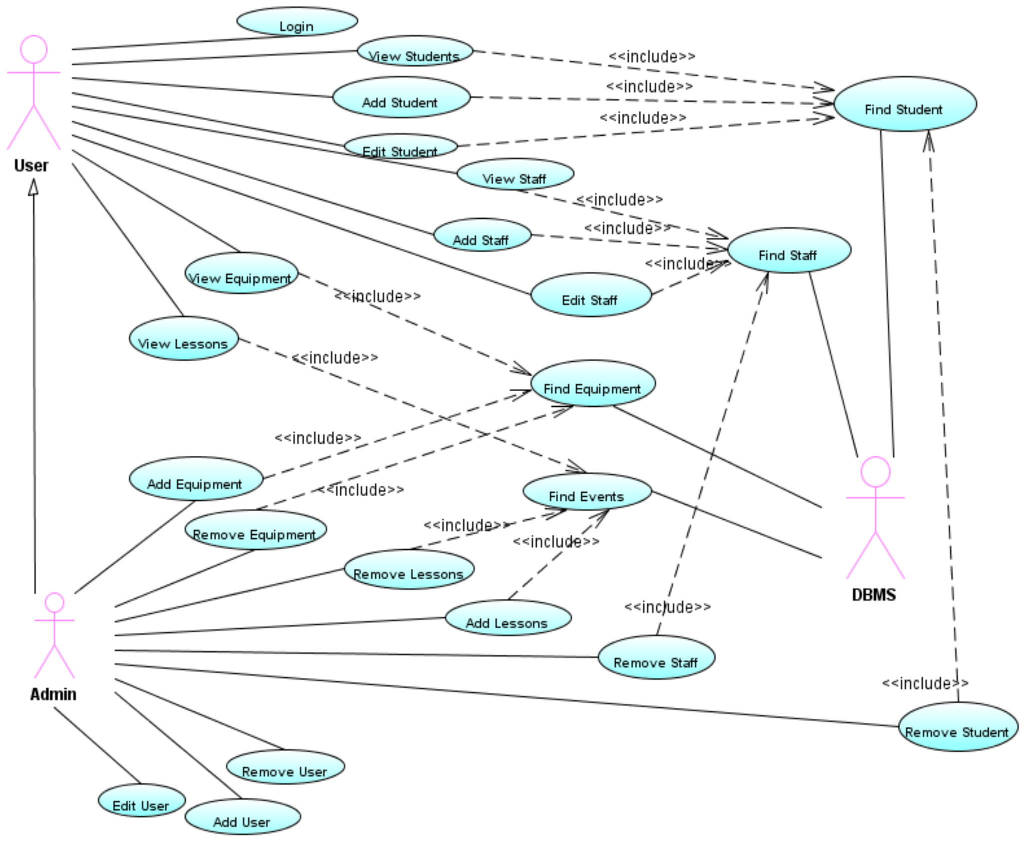


Figure 2 - Actors interaction with system.

This diagram shows a more advanced version of my actors interacting with the system.

I have included in my appendices some use case descriptions that will help me when designing the java application, they can be found in Appendix B.

When creating a timetabling system there are some boolean constraints that need to be considered (Sandhu, 2003).

- Room usage – whether the room is already in use
- Capacity - When adding a student into a room, if the amount of students registered to attend the room is more than the capacity size, then the user will get an error message as the class is full.
- Equipment availability – which equipment is available to which room.
- Student/staff member availability – which lessons they are already attending/teaching and which lessons they are free.
- Room empty – if a room is empty how will we know

Taking these issues into account, I decided that I would deal with the constraints in the java part of my project.

3.2 Non-Functional Requirements

Non- functional requirements refer to the constraints and quality of the system (Malan, 2001). As my project requires the management of what will eventually be a large amount of data, constraints on the amount of actual data displayed on my interface may be beneficial as analysing large amounts of information can be time consuming and confusing.
The main non-functional requirements are:

- Ease of use/ navigation
- Suitable layout
- Updating of software
- System should be fast enough to process large amounts of data

The quality of the system will be dealt with when designing the java application. As I was designing a system that would deal with the timetabling management of the school I needed to also focus on the usability of the system from the user's perspective. This is something that I tried to deal with during the process of designing my java application, but getting the actual system working was my main priority.

4. The Database

The following chapter deals with the design, implementation and testing of the database.

The database is the main part of my project, without a working database I couldn't build an application for it. To be able to fulfil the functional requirements, the database must be designed and implemented to a good standard. To be able to create my application I needed to first design and create the database and check it worked well.

From my research and tutorials I followed I was able to determine what my main approach would be in designing the database. Most of the information that I gathered was from research as I did not have much knowledge of databases before starting this project.

4.1 Design

As discussed in my research (2.2.1) I decided to design my database using conceptual, logical and physical design (Connolly, 2002). By following steps within these concepts I could be sure that I was making the right decisions to create my database.

The design of the database system is discussed in section 4.1.2 the logical and physical design.

4.1.1 Conceptual database design

As discussed in my research, conceptual database design is the first design process I needed to use at the start of designing my database.

I approached this area of design by finding the work processes of the school and the relationships between them, at this stage the model is not concentrated on actual database design but rather the processes involved in the organisation and how they work together.

This knowledge is usually gathered through interviews with potential users of the system but from my experiences of working within a school I was able to come up with the requirements needed myself, which can be found in section 2.3 and chapter 3.

Members of staff needed to use the software for different reasons. The following data requirements list the views of the school seen by the administrators and the general users, they have different uses of the database, and administrators have more administrative rights than general users. For my requirement collection I used a centralised approach to gathering my requirements which meant that the requirements for both administrators and general users are merged into a single set of requirements (Connolly, 2002).

Data Requirements
The following requirements are from the users and administrators view of the school. It details some of the processes that are used in the school currently and that will need to be considered when creating the database.

Lessons:
The school teaches lessons to its students. The information held on a lesson would be which lesson is going to take place, what the start time is of that lesson, and what the end time of it is. Also it will give a date of which day the lesson occurs on. Additional data would be provided of which subject the lesson will be.

Staff:
Members of staff are responsible for the teaching of the school's students. They can teach a maximum of six lessons a day and overall 30 lessons a week. Each teacher belongs to a department which represents the subject they teach. E.g. a dance teacher would belong to the Performing Arts department. The information stored on staff members are: first name, last name, title, department.
Students:
These are pupils within the school. Their ages range from 12-18. To be considered a student they must be part of the school. The information stored on students is: First name, last name, date of birth, address.
Room:
The rooms located in the school in which lessons take place. Information stored on rooms includes room name and the capacity of the room.
Department:
Members of the teaching staff each belong to a department, there are seven overall. The information in department is department name
Subject:
The details of subjects taught are stored. Information stored on subjects is subject name and department name.

Transaction Requirements
Transaction requirements detail the processes used by the administrators or general users to carry out their daily work.

The following transaction requirements I found to be essential to what users needed to be able to do at the school.

Data Entry:
- Enter the details of a new member of staff into the school
- Enter the details of a new student to the school
- Enter a new type of lesson to the school
- Enter a new subject to subjects
- Enter a new department
- Enter new equipment

Data update/deletion:
- Update/delete details of a lesson
- Update/delete details of student
- Update/delete details of a staff member
- Update/delete details of equipment
- Update/delete details of department
- Update/delete details of subject

Data Queries
Examples of queries required by an administrator to follow out their daily tasks:
- List the details of all subjects
- List all students
- List all staff members
- Identify the total number of students/ staff members.
- Identify how many lessons a particular staff member is teaching
- Identify which student is in which lesson

- Identify which lesson a student is in at a particular time.
- Identify which equipment is located in which lesson.

Identifying Entity Types
The next step of the conceptual database design is to identify the main entity type required. I selected the entities from the requirements analysis (Chapter 3), from the processes the users carried out. I decided the following entities were appropriate: Staff, Student, Event, Room, Subject, Department, and Equipment. Table 1 below states the name of each entity, its description and where it can occur.

Table 1 - Main entities and their associated data

Entity Name	**Description**	**Alias**	**Occurrence**
Staff	Describing all teachers within school.	Teachers, employees	Each member of staff teaches events. They belong to one department. Every member of staff teaches a subject.
Student	Describing all students attending the school.	Pupils	Each student attends events.
Event	Describing all events that take place at the school.	Lesson	Each event has one subject. An event is attended by students. An event is taught by teachers.
Room	Describes the rooms that are situated at the school.		A room hosts events. A room also has equipment in it.
Subject	Describes the subjects that are taught at the school		Each subject belongs to a department. Each subject is taught by a teacher.
Department	Describes the departments within the school.		Each department has subjects. The department also has teachers that belong to it.
Equipment	Describes all the equipment at the school.		Equipment can be found in rooms.

Table 2 - Entity relationships between each class

Entity Name	**Multiplicity**	**Relationship**	**Entity Name**	**Multiplicity**
Staff	1..* 1..*	Teaches Belongs	Event Department	1..* 1..*
Student	1..*	Attends	Event	1..*
Room	1..*	Host	Event	1..*
Subject	0..* 1	Taught by At	Staff Event	0..* 1..*
Department	1	Has	Subject	1..*
Event	1..*	Has	Subject	1
Equipment	0..*	Stored in	Room	0..*

The table above shows the relationship each entity has if related to another entity and in which multiplicity the relationship is.

Table 3 - Associated data

Entity	**Associated Data**
Student	Name -first name -last name Age Address -House Number -Post Code
Staff	Name -first name -last name Title Subject Department Phone number
Event	Event name Start time End time Date of event
Room	Room name Room number Capacity
Subject	Subject name
Department	Department name
Equipment	Equipment name

The table above shows the simple associated data that belongs to an entity. This will be useful when planning what will go into the database at a later stage. It begins to identify what information is needed about each entity in order for me to carry out queries and get the information that will actually help me attain the right information.

4.1.2 Logical Database Design and Physical Database Design

As discussed in my research (2.2.1) the logical design is needed to convert the decisions made in conceptual design, into a logical structure that can be implemented into the database. The logical and physical database design are located in the same section as many of the processes involved in each design phase actually overlap and can be quite similar. The physical database design will turn the entities I have chosen in my logical design into tables, my attributes into columns and my relationships into foreign and primary keys.

Every entity in my database needs an entity identifier (Lim, et al., 1993). Entity identifiers need to be meaningless and unrestricted, and so I needed to select something that was appropriate. As an example a telephone number couldn't be used as they often change, also people's names change either due to marriage or divorce. Using a national insurance number could work as it is unique, but in my system this would be restricted by the fact that pupils would not receive their National Insurance numbers until they are sixteen, although it is available to the teachers, this could also be restricted by a teacher who is from another country here on a working visa. As this isn't consistent I needed to have a unique identifier for each entity. From this I made the decision for every entity I had, there would be a unique identifier for it e.g. for my Student entity there is a ‘studentId’ attribute which identifies each student uniquely.

My next task was to decide on the attribute domains and the domain type. A domain decides what the data value is for a specific attribute. A domain type is what the attribute represents and across my attributes there are many different types e.g. names, ages, addresses, dates, time (IBM, 2011). These types need to be represented in the best possible way when creating the database and there is usually a type that is suitable for each attribute. For the date of an event; Date can be used, for the capacity of a room; Integer can be used. Some types of data cannot be specifically defined so for names, Varchar is the best data type that I can use as it is long enough to hold any expected value (Jewett, 2006). The data types I chose can be found in Table 4, pg 21.

I decided that in my database I did not want any multi-valued attributes; this was because using them makes sorting data in a database difficult. To solve my issue of multi-valued attributes I have removed them completely and made additional attributes to resolve this. In my employees entity there was a Phone number attribute, the problem with this is people have more than one number, this was resolved by creating two separate attributes, e.g. home number and work number.

Defining integrity constraints

As mentioned in my research (2.2.1) there are different constraints that need to be considered when creating a database. The five types of integrity constraints that I needed to consider were required data, attribute domain constraints, entity integrity, referential integrity and enterprise constraints.

Required data is where some attributes must hold a value, so nulls cannot always be accepted in a database. Attribute domain constraints defines the data type and length restricting the user to only put in defined values. Entity integrity is defining a primary key to each entity to ensure that when inserting data into a table, no duplicate primary key is created, it also means a primary key cannot hold nulls. Referential integrity ensures that any foreign key declared in a table can only contain values created from a primary key in the parent table (MySQL, 1997-2012).

Enterprise constraints deal with the business rules of the school. An example of this would be that my school didn’t teach more than thirty children in a classroom at any time due to them wanting their pupils to get a better chance to learn.

The table below reflects the decisions I made with my entities and attributes and what I chose as their data types.

Table 4 - Entities, Attributes & data types

Entity Name	**Attributes**	**Description**	**Data Type & Length**	**Null Value**	**Constraints**

Staff	staffId
	
	fName
	lName
	title
	deptID
	
	Home phone
	Mobile Phone
		Uniquely identifies a member of staff
		First name of staff
		Last name of staff
		Title of Staff
		Department staff belongs to
			Varchar (6)
			
			Varchar(45)
			Varchar(45)
			Varchar(10)
			Varchar(3)
			
			Int(11)
			Int(11)
				No
				
				No
				No
				Yes
				Yes
				
				Yes
				Yes	

Student	studentId
	
	First Name
	Last Name
	dob
	address
		Uniquely identifies a student
		First name of student
		Last name of student
		Date of birth
		Home address of student
			Varchar(6)
			
			Varchar(45)
			Varchar(45)
			Date
			Varchar(255)
				No
				
				No
				No
				Yes
				No	

Event	eventId subjectId dateOf startTime endTime	Uniquely identifies an event Identifies subject Date of event Start time of event End time of event	Varchar(10) Varchar(6) Date Time Time	No No No No No	

Subject	subjectId
	
	subName
	deptId
		Uniquely identifies subject
		Name of subject
		Uniquely identifies department id
			Varchar (6)
			
			Varchar(45)
			Varchar(3)
				No
				
				No
				No	

Equipment	rEquipId
	
	equipName
		Uniquely identifies equipment id
		Equipment name
			Varchar(6)
			
			Varchar(45)
				No
				
				No	

Department	deptId
	
	deptName
		Uniquely identifies department
		Name of department
			Varchar(6)
			
			Varchar(45)
				No
				
				No	

Room	roomId
	
	rName
	capacity
		Uniquely identifies room
		Name of room
		Capacity that room can hold
			Varchar(6)
			
			Varchar(45)
			Int(11)
				No
				
				No
				No
					
					<= 30

Table 5 – Primary Key & Foreign Keys

Event (eventId, subjectId, dateOf, startTime, endTime) **Primary Key**: eventId **Foreign Key:** subjectId **references** Subject (subjectId)	**Hostedin** (hostId, roomId, eventId) **Primary Key:** hosted **Foreign Key:** roomId **references** Room(roomId) **Foreign Key:** eventId **references** Event(eventId)
Student (studentId, firstname, lastname, address, DOB) **Primary Key:** studentId	**Attends** (attendsId, studentId, eventId) **Primary Key:** attendsId **Foreign Key:** studentId **references** Student(studentId) **Foreign Key:** eventId **references** Event(eventId)
Staff (staffId, first name, last name, title, deptId) **Primary Key:** staffId **Foreign Key:** deptId **references** Department(deptId)	**Teaches** (teachesId, staffId, eventId) **Primary Key:** attendsId **Foreign Key:** staffId **references** Staff (staffId) **Foreign Key:** eventId **references** Event(eventId)
Subject (subjectId, subName, deptId*) **Primary Key:** subjectId **Foreign Key:** deptId **references** Department (deptId)	**Expertise** (expertiseId, staffId*, subjectId*) **Primary Key:** expertiseId **Foreign Key:** staffId references Staff (staffId) **Foreign Key:** subjectId references Subject (subjectId)
Room (roomId, roomName, Capacity) **Primary Key:** roomId	**Department** (deptId, deptName) **Primary Key:** deptId
Equipment(equipId, equipName) **Primary Key:** equipId	**EquipinRoom** (equipId, eventId, roomId) **Primary Key:** equipId **Foreign Key:** eventId **references** Event(eventId) **Foreign Key:** roomId **references** Room (roomId)

The table above reflects my decisions for use of my entities and their relationships with other entities; it details what the primary and foreign keys are and what tables they are referenced to. It extends on from table 2 as it identifies what my primary and foreign keys are going to be in my database, which hadn't yet been decided in the previous table.

Logical Conceptual Data Model

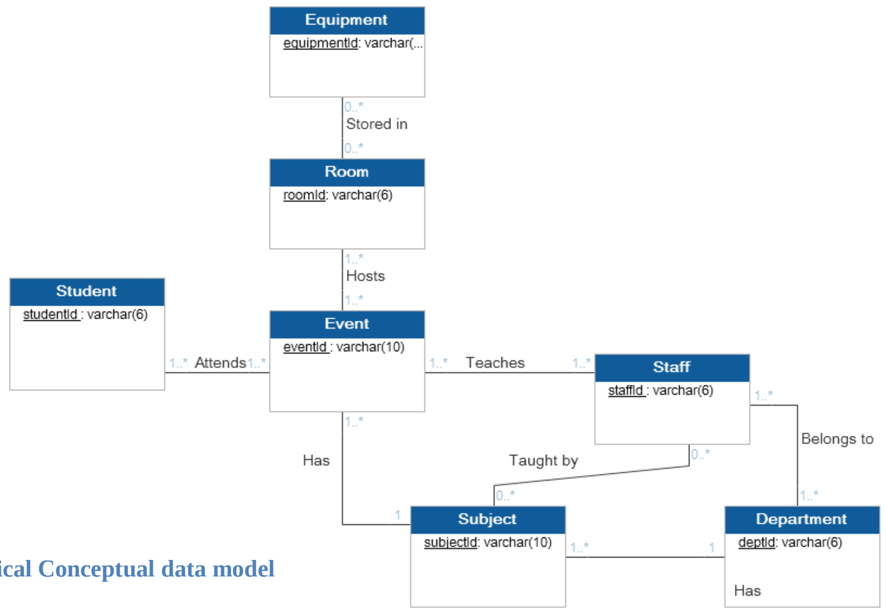


Figure 3 - Logical Conceptual data model

At this stage I realised that I needed to remove the many-to-many relationships that I had within my project. The reason being is that many-to-many relationships do not actually work well and most relational database applications only support one-to-many relationships (Database Dev, 2003). To remove my many-to-many relationships from my diagram I had to create a link table between each many-to-many relationship I had in my diagram. I chose names appropriate to the relationship to do this. The diagram below shows the logical conceptual data model with features not compatible with the relational model removed.

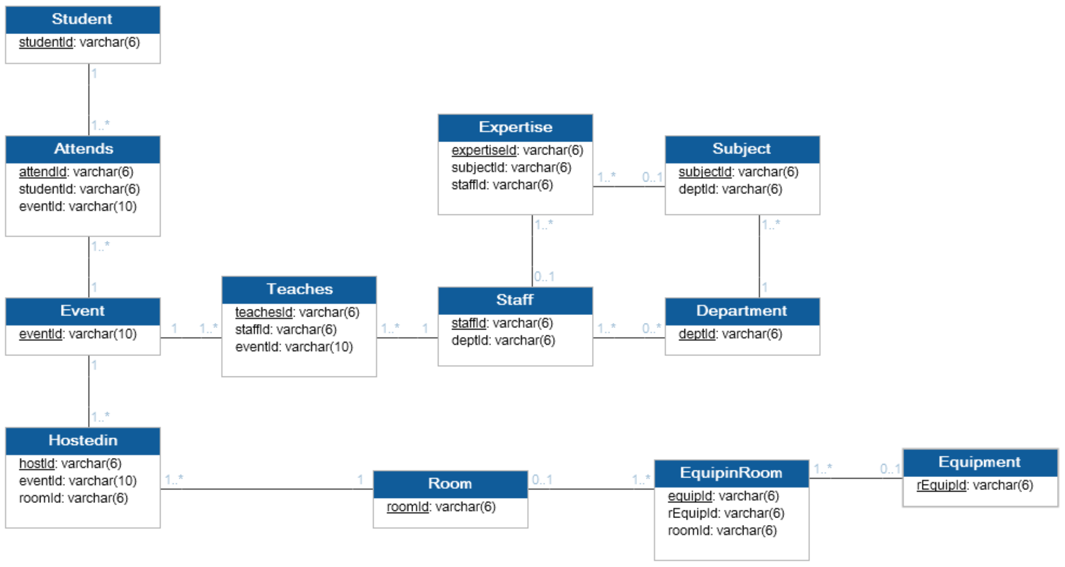


Figure 4 - The final logical conceptual data model

The diagram introduces my link entities, which are:

- Teaches
- EquipinRoom
- Expertise
- Hostedin
- Attends

Entities:
The main entities of my database were: Staff, Student, Event, Room, Subject, Department, and Equipment.
Many of these entities had many to many relationships with surrounding tables. As I discussed in my research (2.2.1) a link table is needed to manage many –to-many relationships, by creating a link table the relationship was changed from many-to-many to one-to-many which is ideal for a relational database as the majority of DBMS do not support these relationships. The following table shows the relationships my entities have with each other in detail describing what they do and how they work with the entity they have a relationship with.

Table 6 - Relationships between entities

Entities	**Description of entity**	**Relationships with other entities.**
Staff	Stores personal information regarding teachers, also affiliates staff with the department they belong to.	Expertise – stores the information of staff member and also of the subject they teach.
		Teaches – stores the information of staff member and the events they teach.
Student	Stores all personal information of students including their home address and date of birth.	Attends – stores the information of student and the events that they are registered to.
Event	Stores information regarding event, date of event, time event starts and time it ends. Also holds subject id as a foreign key.	Attends – keeps information regarding which studentID is registered to a particular eventID.
		Teaches – keeps information of eventId and staffId which is teaching that particular event.
		Subject is a foreign key within event, needed for the relationship between an event and subject.
		Hostedin – links information of events to rooms.
Room	Stores information regarding room name, room capacity	Equipinroom – links information regarding equipment to roomID.
		Hostedin – links information of the eventid that that is hosted in the particular room with roomed.
Subject	Holds details about the subject. It also has department as a foreign key.	Department – links information of subject with a department ID. E.g. History belongs to humanities department.
Department	Holds information regarding departments.	Staff – provides information regarding which department staff member belongs to.
		Subject – provides information regarding which subject belongs to a certain department.
Equipment	Holds information of what equipment is available in the school.	EquipinRoom – stores details of which equipment belongs in which room.

Relationships
The next section shows the relationship between an entity and any other entities they may interact with. I have described how each one interacts with the other.

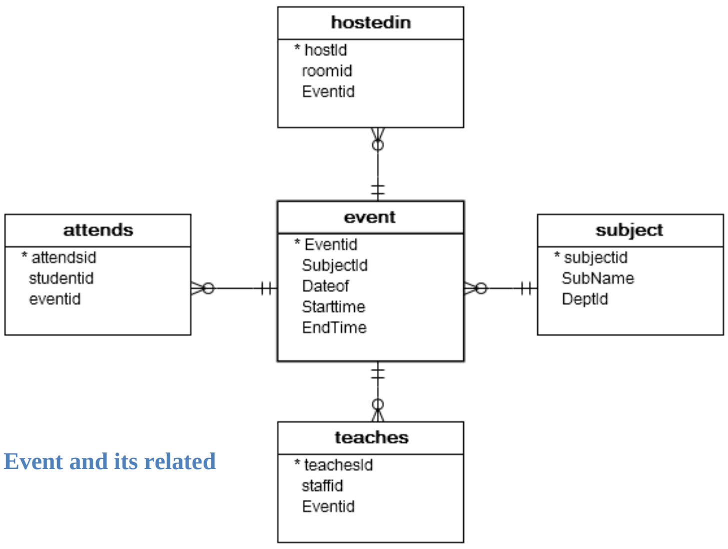


Figure 5 - Event and its related Entities

This ER diagram shows the relationships ‘Event’ has with other tables. ‘Event’ stores information regarding date of event, time event starts and time event ends. It is connected to three tables that each have ‘eventid’ referenced as a foreign key, which are the link tables ‘hostedin’, ‘attends’ and ‘teaches’ these allow ‘Event’ to have a relationship with another table that is one-to-many, rather than many-to-many. ‘Subjectid’ is referenced in ‘Event’ as a foreign key, which is referenced to the ‘Subject’ table.

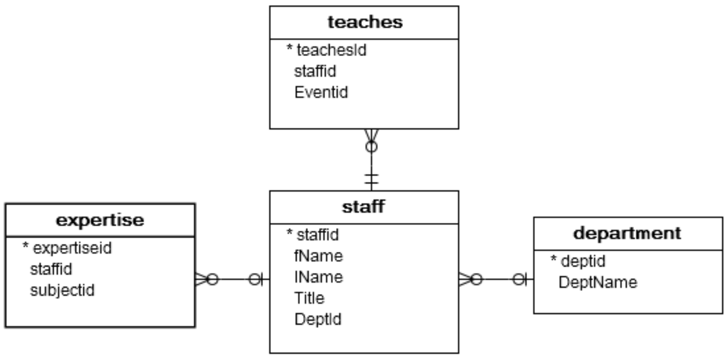


Figure 6 - Staff and its related entities.

‘Staff’ stores important information regarding its staff members. ‘Expertise’ and ‘Teaches’ both reference ‘StaffId’ as a foreign key in their table, these allow ‘Staff’ to have a relationship with another table that is one-to-many, rather than many-to-many.   ‘Deptid’ is referenced in ‘Event’ as a foreign key, which is referenced to the Department table.

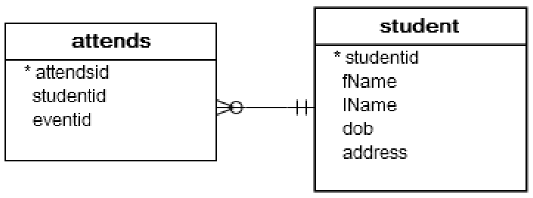


Figure 7 - Student and its related entity

‘Student’ stores all relevant information regarding student’s personal details. Student has a one-to-many relationship with the link entity ‘Attends’ which allows the student to be paired with events. ‘Studentid’ is a foreign key in the ‘attends’ table which is referenced to ‘Student’.

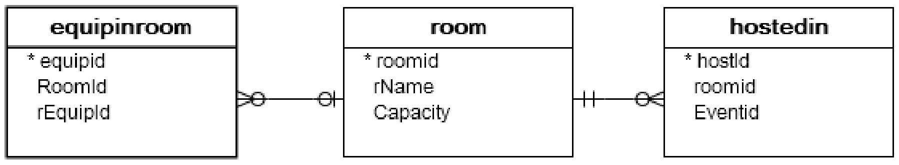


Figure 8- Room and its related entities

‘Staff’ stores important information regarding its staff members. ‘hostedin’ and ‘equipinroom’ both reference ‘roomId’ as a foreign key in their table, these allow ‘Room’ to have a relationship with another table that is one-to-many, rather than many-to-many.

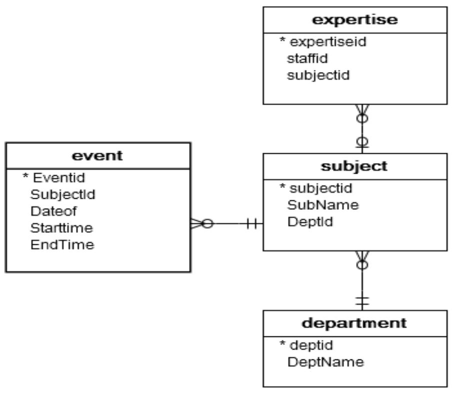


Figure 9 - Subject and its related entities

‘Subject’ stores all information regarding subjects. It has a one-to-many relationship with ‘event’ where its primary key is located in events table as a foreign key. Subject has a many-to-one relationship with ‘department’ which is referenced in its table as a foreign key. ‘Expertise’ references ‘roomid’ as a foreign key in its table.

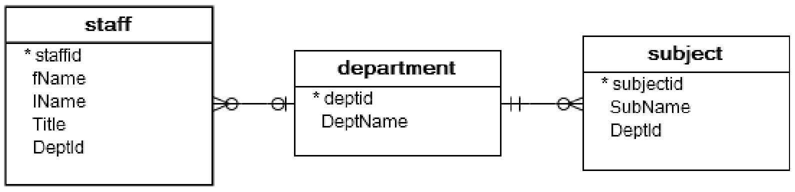


Figure 10 - Department and its related entities

The ‘Department’ entity stores information regarding departments. It is referenced as a foreign key in both Staff and Subject tables.

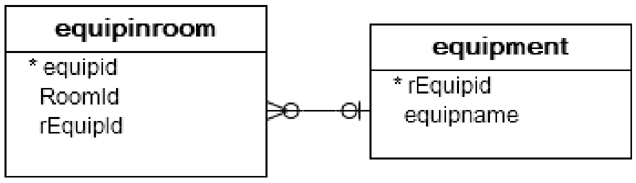


Figure 11 – Equipment and its related entity.

‘Equipment’ stores information regarding what equipment is within the school. It has a one-to-many relationship with its link table ‘equipinroom’ which stores what equipment is in which room.

Details of how they’re all referenced can be found in table 5 on page 24.

4.2 Implementation

The following section discusses the decisions I made when implementing my database and any problems I encountered during the implementation of it.

I took the decision to continually test my database whilst creating it, which is why some examples of testing can be found in my implementation section. As it was my first time creating a database I needed to check that what I was doing was correct.

4.2.1 Creating my database

In this section I describe how I created my database in MySQL.

As discussed in my research to create my database I used Data definition language (DDL), this syntax is used to define the database and create entities and input data into my database. The first DDL I used was CREATE, which deals with the creation of my database. The code I used was “`CREATE database IF NOT EXISTS Project`”. By using IF NOT EXISTS MYSQL checks whether there is another database already in use with that name, if there is it will bring up an error message, if not the database will be successfully created.

I have given two examples of how I made two of my tables and the creation code for the remaining tables can be found in Appendix C.

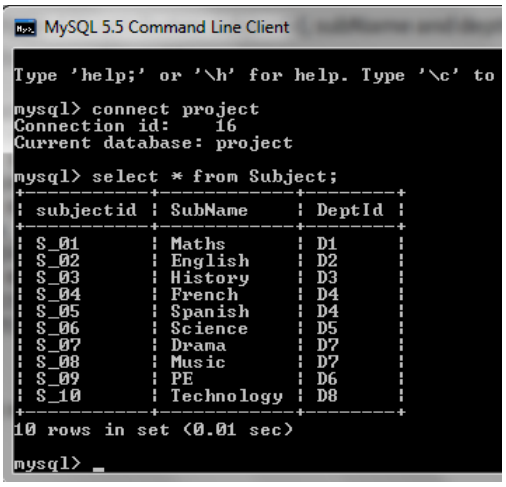
The code below describes how I made my Department table; it represents the creation of my Department entity and its attributes: ‘deptId’, ‘deptName’.

```
CREATE TABLE IF NOT EXISTS Department (
  deptId VARCHAR(6) NOT NULL DEFAULT,
  deptName VARCHAR(45) NOT NULL ,
  PRIMARY KEY (deptId))
```

The code below describes how I created my Subject table; it represents the creation of my Subject entity and its attributes: ‘subjectId’, ‘subName’ and ‘deptId’. ‘deptId’ is a foreign key referenced from another table.

```
CREATE TABLE IF NOT EXISTS  Subject (
  subjectId VARCHAR(6) NOT NULL DEFAULT ,
  subName VARCHAR(45) NOT NULL ,
  deptId VARCHAR(3) NOT NULL ,
  PRIMARY KEY (subjectid),
  INDEX fk_SubjectDept (deptId ASC),
  CONSTRAINT fk_SubjectDept
  FOREIGN KEY (deptId)
  REFERENCES  department (deptid )
  ON DELETE RESTRICT)
```

To test that my table had been constructed properly I performed the query `'SELECT * FROM Subject';` the following screen shot shows worked.



```
Type 'help;' or '\h' for help. Type '\c' to

mysql> connect project
Connection id:    16
Current database: project

mysql> select * from Subject;
+-----------+------------+--------+
| subjectid | SubName    | DeptId |
+-----------+------------+--------+
| S_01      | Maths      | D1     |
| S_02      | English    | D2     |
| S_03      | History    | D3     |
| S_04      | French     | D4     |
| S_05      | Spanish    | D4     |
| S_06      | Science    | D5     |
| S_07      | Drama      | D7     |
| S_08      | Music      | D7     |
| S_09      | PE         | D6     |
| S_10      | Technology | D8     |
+-----------+------------+--------+
10 rows in set (0.01 sec)

mysql> _
```

Figure 12 – Screenshot of query

4.3 Testing

Testing of a database is called back end testing; this is because we are testing without the use of a user interface. Structural and functional testing are the most effective ways of backend testing (Ambler, 2010). The structural part of testing focuses on database design, tables, columns, column types, keys and rules. Functional testing focuses on the functionality and features of backend and deals with data integrity and consistency.

To test my database correctly I needed to check that my entire physical database matched the design of my database and that the data types I had chosen had been correctly implemented. I also needed to check that I could perform use cases I had created in my functional requirements in section 3.1 with the database I created.

When testing my database I needed to check that my database had data integrity and data validity. Data integrity checks the accuracy of my data and to check the integrity I needed to create, modify and delete information from my tables.

To check data validity I needed to check that the correct data appears in the correct fields e.g. a phone number will be numerical data. It will also check whether my foreign constraints are assigned correctly and that a row cannot be deleted if it belongs to another table. I also needed to check that any values that are supposed to be unique cannot be duplicated.

To test my database and verify whether my use cases had been fulfilled I used the following methods:

- Inserting data into the database
- Correct data types used and testing incorrect values in columns
- Foreign keys referenced correctly
- Deleting and updating data in the database
- Creating advanced queries for the database

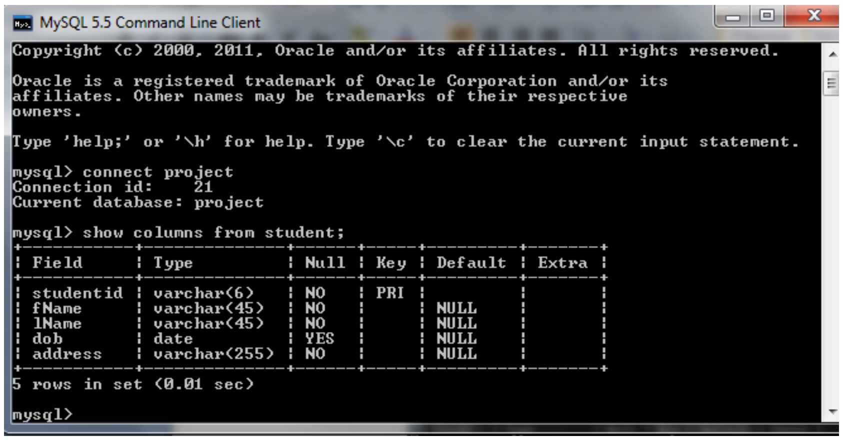
Test data:

For my database I chose to use test data, this is because it was simpler to create my own details and to create it in a way that I could understand, this way I could check the results that came out would be as I expected. By producing my own data I was able to specifically create it so that I could test my database and check it was working properly. Also I wouldn't have been able to retrieve this data from my university or the school due to the Data Protection Act (UK, 1998).

As discussed in my research (2.2.1) to manipulate and retrieve data from a database data manipulation language (DML) is used. The DML language allowed me to properly analyse the data I put into my database so I can check that it is performing as it should.

4.3.1 Inserting Data into database

To insert values into my database I first needed to check what my attributes were to make sure I was inserting the correct details. By typing 'SHOW COLUMNS FROM Student' I was able to check my attributes before I entered any data.



```
Copyright (c) 2000, 2011, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> connect project
Connection id:    21
Current database: project

mysql> show columns from student;
+-----------+--------------+------+-----+---------+-------+
| Field     | Type         | Null | Key | Default | Extra |
+-----------+--------------+------+-----+---------+-------+
| studentid | varchar(6)   | NO   | PRI |         |       |
| fName     | varchar(45)  | NO   |     | NULL    |       |
| lName     | varchar(45)  | NO   |     | NULL    |       |
| dob       | date         | YES  |     | NULL    |       |
| address   | varchar(255) | NO   |     | NULL    |       |
+-----------+--------------+------+-----+---------+-------+
5 rows in set (0.01 sec)

mysql>
```

Figure 13 – Screenshot of query

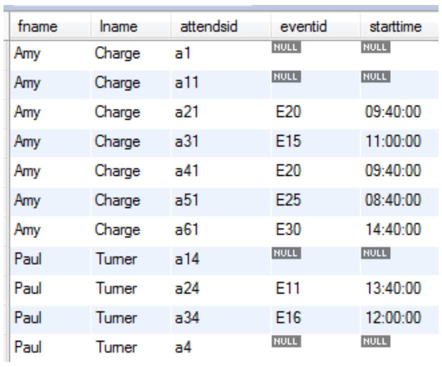
When I first started creating my database I made the decision to pass a CSV file containing data in each column that represented 'studentid', 'fname', 'lname', 'dob' and 'address' which allowed me to insert a large amount of data into my database. Creating a CSV file makes inserting data easier as all that is needed is a spreadsheet filled with the data.
The following code shows how I did this:

```
LOAD DATA INFILE "filename.csv" INTO TABLE tablename
```

```
COLUMNS
TERMINATED BY ',';
```

When performing a query to return students with the list of events that they had been registered to I came across a problem in my data. In this query ‘S1’ (Amy) attends ‘A1’, but in my results it returned null on ‘eventId’ and ‘start time’, at first I thought it was something that could be overlooked, but when I started having issues with the results of queries coming out wrong, I went through and highlighted the records I was having problems with I noticed that ‘attendsId’ A1-A18, the results were all returning null and from A19-A62 they all worked and returned the correct values.

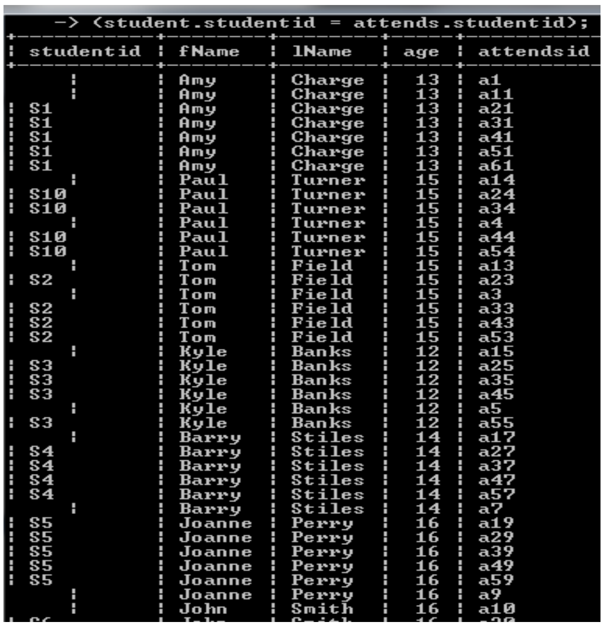
The following figure shows how the errors were shown.



fname	lname	attendsid	eventid	starttime
Amy	Charge	a1	NULL	NULL
Amy	Charge	a11	NULL	NULL
Amy	Charge	a21	E20	09:40:00
Amy	Charge	a31	E15	11:00:00
Amy	Charge	a41	E20	09:40:00
Amy	Charge	a51	E25	08:40:00
Amy	Charge	a61	E30	14:40:00
Paul	Turner	a14	NULL	NULL
Paul	Turner	a24	E11	13:40:00
Paul	Turner	a34	E16	12:00:00
Paul	Turner	a4	NULL	NULL

Figure 14 – Screenshot of query

When I did a query to find the results of what had been added to the tables I found that my ‘Student’ table returned strange characters under the ‘studentId’ (as shown in the screen shot below). I then noticed the strange characters were appearing on A1-A18, in my ‘attendsId’, which were the values returning Null in my previous query.



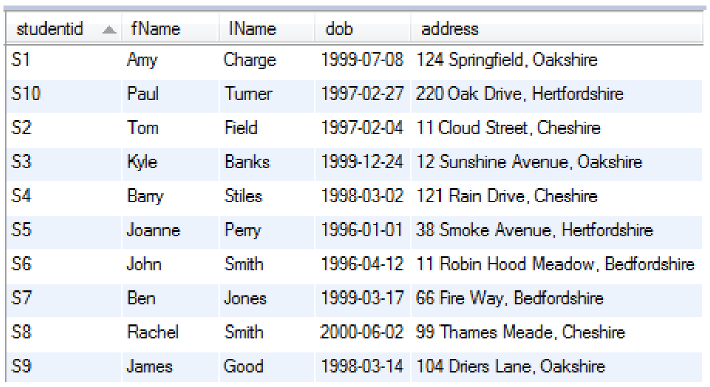
```
    -> (student.studentid = attends.studentid);
+-----------+--------+---------+-----+----------
| studentid | fName  | lName   | age | attendsid
+-----------+--------+---------+-----+----------
       |    | Amy    | Charge  |  13 | a1
       |    | Amy    | Charge  |  13 | a11
| S1        | Amy    | Charge  |  13 | a21
| S1        | Amy    | Charge  |  13 | a31
| S1        | Amy    | Charge  |  13 | a41
| S1        | Amy    | Charge  |  13 | a51
| S1        | Amy    | Charge  |  13 | a61
       |    | Paul   | Turner  |  15 | a14
| S10       | Paul   | Turner  |  15 | a24
| S10       | Paul   | Turner  |  15 | a34
       |    | Paul   | Turner  |  15 | a4
| S10       | Paul   | Turner  |  15 | a44
| S10       | Paul   | Turner  |  15 | a54
       |    | Tom    | Field   |  15 | a13
| S2        | Tom    | Field   |  15 | a23
       |    | Tom    | Field   |  15 | a3
| S2        | Tom    | Field   |  15 | a33
| S2        | Tom    | Field   |  15 | a43
| S2        | Tom    | Field   |  15 | a53
       |    | Kyle   | Banks   |  12 | a15
| S3        | Kyle   | Banks   |  12 | a25
| S3        | Kyle   | Banks   |  12 | a35
| S3        | Kyle   | Banks   |  12 | a45
       |    | Kyle   | Banks   |  12 | a5
| S3        | Kyle   | Banks   |  12 | a55
       |    | Barry  | Stiles  |  14 | a17
| S4        | Barry  | Stiles  |  14 | a27
| S4        | Barry  | Stiles  |  14 | a37
| S4        | Barry  | Stiles  |  14 | a47
| S4        | Barry  | Stiles  |  14 | a57
       |    | Barry  | Stiles  |  14 | a7
| S5        | Joanne | Perry   |  16 | a19
| S5        | Joanne | Perry   |  16 | a29
| S5        | Joanne | Perry   |  16 | a39
| S5        | Joanne | Perry   |  16 | a49
| S5        | Joanne | Perry   |  16 | a59
       |    | Joanne | Perry   |  16 | a9
       |    | John   | Smith   |  16 | a10
```

Figure 15 – Screenshot of problem

It did take some testing to discover it was the way I had been inserting data into my tables which was had caused the problem, the CSV file I had used and the code used to insert the CSV file into the database was producing abnormal characters in some of my tables. After this issue I decided that the best way for me to insert data would be manually into the tables as I knew that it wouldn't have the same problems as I'd had with the CSV file.

To add data into my Staff table, I needed to enter the following values:
'staffId', 'fName', 'lName', 'title', 'deptId'
For my 'Student' table I added the values in with the following method.

```
INSERT INTO Student VALUES
  ('S1','Amy','Charge','1999-07-08','124 Springfield,
  Oakshire'),
  ('S2','Tom','Field','1997-02-04', '11 Cloud Street,
  Cheshire'),
  ('S3','Kyle','Banks','1999-12-24','12 Sunshine Avenue,
  Oakshire'),
  ('S4','Barry','Stiles','1998-03-02','121 Rain Drive,
  Cheshire'),
  ('S5','Joanne','Perry','1996-01-01', '38 Smoke Avenue,
  Hertfordshire'),
  ('S6','John','Smith','1996-04-12','11 Robin Hood Meadow,
  Bedfordshire'),
   ('S7','Ben' ,'Jones',' 1999-03-17','66 Fire Way,
  Bedfordshire'),
   ('S8','Rachel', 'Smith','2000-06-02','99 Thames Meade,
  Cheshire'),
   ('S9','James', 'Good',' 1998-03-14','104 Driers Lane,
  Oakshire'),
   ('S10','Paul', 'Turner','1997-02-27','220 Oak Drive,
  Hertfordshire');
```



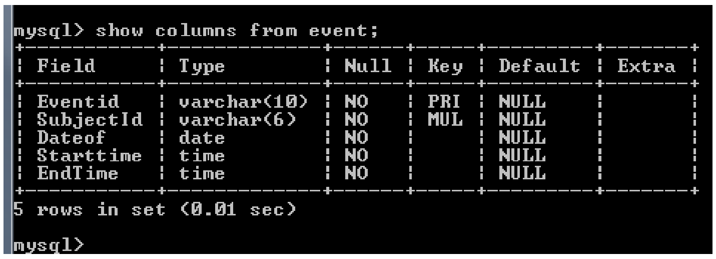
studentid	fName	lName	dob	address
S1	Amy	Charge	1999-07-08	124 Springfield, Oakshire
S10	Paul	Turner	1997-02-27	220 Oak Drive, Hertfordshire
S2	Tom	Field	1997-02-04	11 Cloud Street, Cheshire
S3	Kyle	Banks	1999-12-24	12 Sunshine Avenue, Oakshire
S4	Barry	Stiles	1998-03-02	121 Rain Drive, Cheshire
S5	Joanne	Perry	1996-01-01	38 Smoke Avenue, Hertfordshire
S6	John	Smith	1996-04-12	11 Robin Hood Meadow, Bedfordshire
S7	Ben	Jones	1999-03-17	66 Fire Way, Bedfordshire
S8	Rachel	Smith	2000-06-02	99 Thames Meade, Cheshire
S9	James	Good	1998-03-14	104 Driers Lane, Oakshire

Figure 16 – Screenshot of query

The table above shows that my data was added successfully into the table.
I also inserted data into all my other tables using the INSERT method which can be found in Appendix D.

4.3.2 Data values

To check the data values and that my tables were correct and matched my designs I needed to perform the query “SHOW COLUMNS FROM table_name”, from the query result I was able to check against my design whether my values were correct. Once I had checked this and if I discovered any problems I was able to correct them.   An example of how I did this appears in the screenshot below, by typing ‘SHOW COLUMNS FROM Event; I was given a list of the attributes and their types and whether they were a primary key. With this information and the table listing my data types in table 4, section 4.1.2, I was able to see whether I had inputted the correct data values.



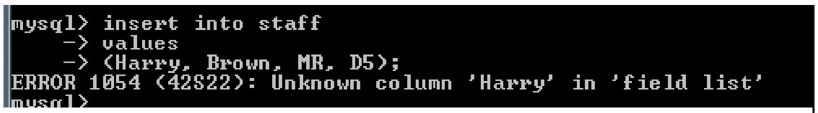
```
mysql> show columns from event;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| Eventid   | varchar(10) | NO   | PRI | NULL    |       |
| SubjectId | varchar(6)  | NO   | MUL | NULL    |       |
| Dateof    | date        | NO   |     | NULL    |       |
| Starttime | time        | NO   |     | NULL    |       |
| EndTime   | time        | NO   |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
5 rows in set (0.01 sec)

mysql>
```

Figure 17 – Screenshot of query

Inserting a null value into a column that does not allow nulls:

If I tried adding a value in with a value missing e.g. ‘staffID’ the database produced the following error “Unknown column ‘Harry’ in field list” which can be seen in she screenshot below. As the ‘staffId’ value cannot be null, a value must be inserted into the ‘staffId’ column. This is the error response I expected as a user should not be able to add null values into a column that were specified as ‘Not Null’ when I created the table.



```
mysql> insert into staff
    -> values
    -> (Harry, Brown, MR, D5);
ERROR 1054 (42S22): Unknown column 'Harry' in 'field list'
mysql>
```

Figure 18 – Screenshot of query

4.3.3 Foreign Keys

Another check I needed to perform was making sure my foreign keys were referenced to the right tables. I did initially have problems with the way I had set up my foreign key constraints, which meant when I performed ‘DROP table_name’ to check my tables couldn’t be removed from the database, unfortunately they did actually end up being deleted. This meant that my database didn’t have referential integrity, a table that holds values about another table (foreign key) should not be able to be easily deleted from the database since this can lead to errors across the system. I used the ALTER DML to change my foreign key constraints. The tables I noticed I had problems with were my link tables, which were the tables that had multiple foreign key references. The code below shows how I changed my table so that the correct foreign key constraints were added to the ‘Attends’ table. The code for the other tables I had issues with can be found in Appendix E.

```
ALTER TABLE attends ADD CONSTRAINT FK_AttendsStudent
FOREIGN KEY (studentId) REFERENCES Student (studentid) ON
DELETE RESTRICT,
ADD CONSTRAINT FK_AttendsEvent FOREIGN KEY (eventId)
REFERENCES event (eventid) ON DELETE RESTRICT;
```

The code above correctly references the foreign key in the right table, using ON DELETE RESTRICT means that you cannot delete a parent row if a child row exists that references the value for that parent row (MySQL, 1997-2012). If the parent row has no referencing child rows, then the parent row can be deleted otherwise if it is set to ON DELETE CASCADE, which when the row in the parent table is deleted, the corresponding matching row in the child table will be deleted, this can be dangerous in a database with link tables, just by deleting one record, it could wipe data from other tables when it wasn't even intended.

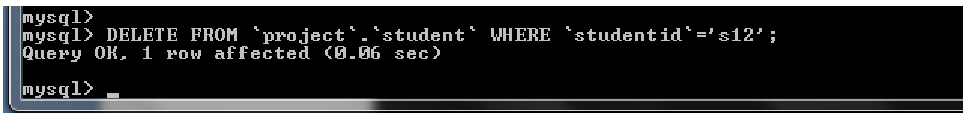
4.3.4 Data Manipulation

As discussed in my research (2.2.1) the data manipulation language allows us to use the data in the database to our advantage, it includes the Select, Delete, Update and Insert statements. In the following section I discuss what I did with the DML syntax to test my database and to also generate queries that can return results that will be of use to my timetabling system.

Deleting Data

To check data will be deleted from my database without any problems I used the following syntax.

```
DELETE FROM `project`.`student` WHERE `studentid`='s12'
```



```
mysql>
mysql> DELETE FROM `project`.`student` WHERE `studentid`='s12';
Query OK, 1 row affected (0.06 sec)

mysql> _
```

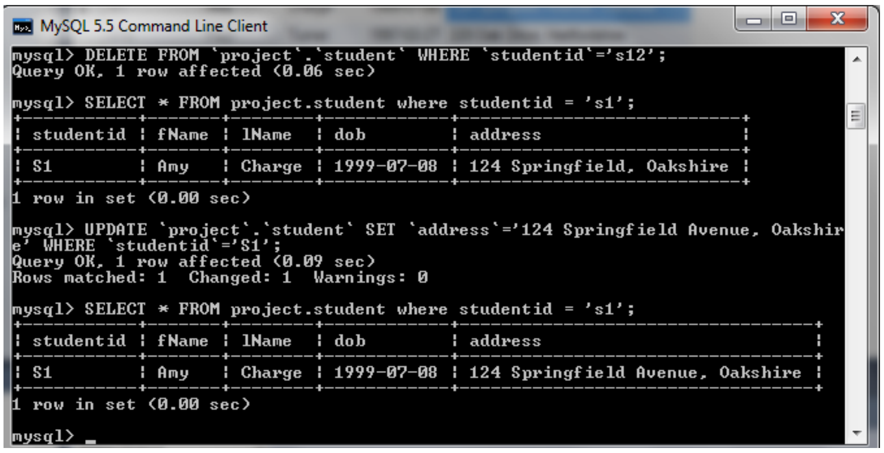
Figure 19 – Screenshot of query

If I got the message that appeared in the screenshot above, I knew I had been successful in deleting my data.

Updating Data:

To check my rows could be updated I performed the following query.

```
UPDATE `project`.`student` SET `address`='124 Springfield
Avenue, Oakshire' WHERE `studentid`='S1';
```



```
mysql> DELETE FROM `project`.`student` WHERE `studentid`='s12';
Query OK, 1 row affected (0.06 sec)

mysql> SELECT * FROM project.student where studentid = 's1';
+-----------+-------+--------+------------+----------------------------+
| studentid | fName | lName  | dob        | address                    |
+-----------+-------+--------+------------+----------------------------+
| S1        | Amy   | Charge | 1999-07-08 | 124 Springfield, Oakshire  |
+-----------+-------+--------+------------+----------------------------+
1 row in set (0.00 sec)

mysql> UPDATE `project`.`student` SET `address`='124 Springfield Avenue, Oakshir
e' WHERE `studentid`='S1';
Query OK, 1 row affected (0.09 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM project.student where studentid = 's1';
+-----------+-------+--------+------------+----------------------------------+
| studentid | fName | lName  | dob        | address                          |
+-----------+-------+--------+------------+----------------------------------+
| S1        | Amy   | Charge | 1999-07-08 | 124 Springfield Avenue, Oakshire |
+-----------+-------+--------+------------+----------------------------------+
1 row in set (0.00 sec)

mysql> _
```

Figure 20 – Screenshot of query

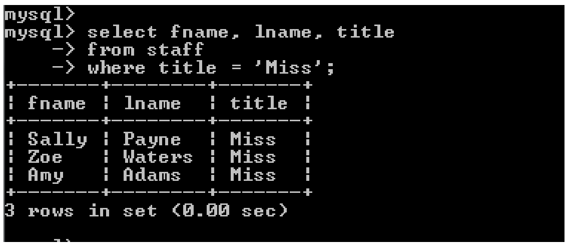
The screenshot above shows that I was successful in updating the record in the table as the address changes from '124 Springfield, Oakshire' to '124 Springfield Avenue, Oakshire'.

Selecting Data:

The 'Select' method can return useful information from my database; there are many different methods that can be used when creating Select statements. By using the statement WHERE in the syntax it allows for many different queries to be produced.

I tested the where statement by creating a query that returned all the staff members with titles that were 'Miss'. The following screenshot shows the results that were produced.

```
select fname, lname, title
from staff
where title = 'Miss'
```



```
mysql>
mysql> select fname, lname, title
    -> from staff
    -> where title = 'Miss';
+-------+--------+-------+
| fname | lname  | title |
+-------+--------+-------+
| Sally | Payne  | Miss  |
| Zoe   | Waters | Miss  |
| Amy   | Adams  | Miss  |
+-------+--------+-------+
3 rows in set (0.00 sec)
```

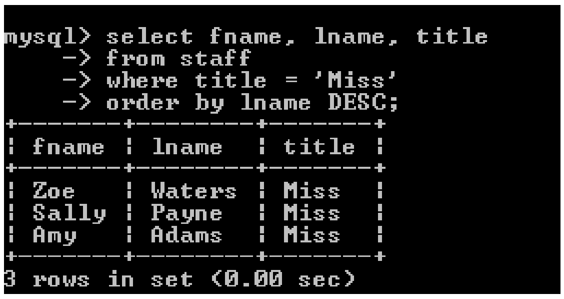
Figure 21 – Screenshot of query

This same query can then have a 'Order by' clause added to it which will allow me to order my results, for a larger table this is extremely useful as you can search alphabetically for something rather than trying to see through mass amounts of data.

The following code is an example of how an order by clause can be created.

```
Select fname, lname, title
From staff
Where title = 'Miss'
Order by lname DESC;
```

The picture below shows it was successful as the last names appear in descending order.



```
mysql> select fname, lname, title
    -> from staff
    -> where title = 'Miss'
    -> order by lname DESC;
+-------+--------+-------+
| fname | lname  | title |
+-------+--------+-------+
| Zoe   | Waters | Miss  |
| Sally | Payne  | Miss  |
| Amy   | Adams  | Miss  |
+-------+--------+-------+
3 rows in set (0.00 sec)
```

Figure 22 – Screenshot of query

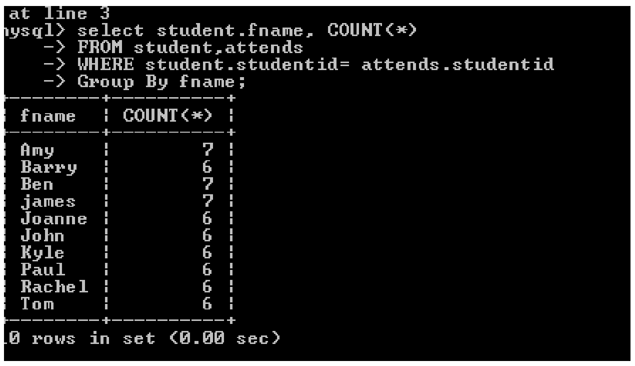
Group by clause:

A query that includes the group by clause is called a grouped query (IBM, 2005). It groups queries from the select table a produces a single row for each group, summarising the data in that row (Connolly, 2002). The group by clause is useful for querying how much of something there is, it is mostly used for working out aggregate functions which is where multiple rows are joined together to form something with more significant meaning, such as working out the overall budget of a department (PostgreSQL, 1996-2012).

The following code is for selecting students and the amount of lessons they are currently assigned to, this is a good query that I can use when checking that students are signed up to the right amount of classes and when the system is completed they should all be registered to thirty lessons.

```
SELECT student.fname, COUNT(*)
FROM student,attends
WHERE student.studentid= attends.studentid
GROUP BY fname;
```

The screenshot below shows the results of the query and how many lessons each student is registered to.



```
 at line 3
ysql> select student.fname, COUNT(*)
    -> FROM student,attends
    -> WHERE student.studentid= attends.studentid
    -> Group By fname;
+--------+----------+
| fname  | COUNT(*) |
+--------+----------+
| Amy    |        7 |
| Barry  |        6 |
| Ben    |        7 |
| james  |        7 |
| Joanne |        6 |
| John   |        6 |
| Kyle   |        6 |
| Paul   |        6 |
| Rachel |        6 |
| Tom    |        6 |
+--------+----------+
0 rows in set (0.00 sec)
```

Figure 23 – Screenshot of query

4.3.5 Queries

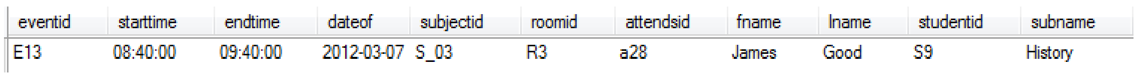
Testing my database with queries helped me understand that the relationships between the tables worked correctly. I have listed all the queries I have done on my database in Appendix F. The queries which are listed below are my more advanced queries which help me achieve some of the aims of my project. My advanced queries involved multi-table joins which combine columns from multiple tables into a results table, this is called a join query.

Searching for students

The following query allowed me to search for a student by their last name between a certain time e.g. 08:40 and 10:30 and find which room the student is located in and which subject lesson they are currently in.

```
SELECT a.*, b.roomid, c.attendsid, d.*, e.subname FROM (
    SELECT eventid, starttime, endtime, dateof, subjectid FROM
    event
    WHERE starttime >= '084000'AND endtime <= '103000'
    ) a
    JOIN (
    SELECT roomid,eventid FROM hostedin
    ) b
    ON a.eventid = b.eventid
    JOIN (
    SELECT eventid, studentid, attendsid FROM attends
    ) c
    ON c.eventid = b.eventid
    JOIN(
    SELECT fname, lname, studentid from student WHERE lname
    ='good'
    ) d
    ON d.studentid = c.studentid
    JOIN(
    SELECT SubName, subjectid from subject
    ) e
    ON e.subjectid = a.subjectid
```

This query joins five of my tables together which are: ‘Event’, ‘Hostedin’, ‘Attends’, ‘Student’ and ‘Subject’. The query results are shown below. The result shows that I have managed to provide the correct data in each table and that the relationships between my tables work as I am able to get a correct response back.



```
eventid  starttime  endtime   dateof      subjectid  roomid  attendsid  fname  lname  studentid  subname
E13      08:40:00   09:40:00  2012-03-07  S_03       R3      a28        James  Good   S9         History
```

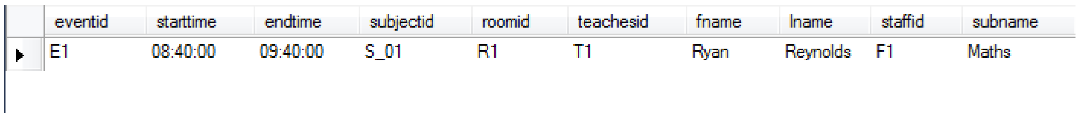
Figure 24 – Screenshot of query

Searching for staff members

I also did a similar query to search for staff members. This query allows me to search for a member of staff by their last name and between a certain time e.g. 08:40 and 10:30 to find which room the staff member is located in and which subject lesson they are currently teaching. This query joins five of my tables together which are: ‘Event’, ‘Hostedin’, ‘Teaches’, ‘Student’ and ‘Subject’.

```
SELECT a.*, b.roomid, c.teachesid, d.*, e.subname FROM (
    SELECT eventid, starttime, endtime, dateof, subjectid FROM
    event
    WHERE starttime >= '084000'AND endtime <= '103000'
    ) a
    JOIN (
    SELECT roomid,eventid FROM hostedin
    ) b
    ON a.eventid = b.eventid
    JOIN (
    SELECT eventid, staffid, teachesid FROM teaches
    ) c
    ON c.eventid = b.eventid
    JOIN(
    SELECT fname, lname, staffid from staff WHERE lname =
      'Ryan'
    ) d
    ON d.staffid = c.staffid
    JOIN(
    SELECT SubName, subjectid from subject
    )e
    ON e.subjectid = a.subjectid
```

The query results are shown below.



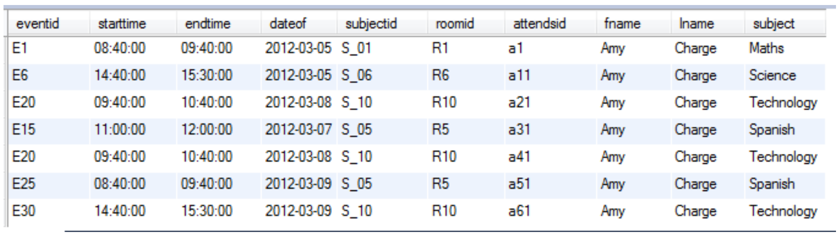
eventid	starttime	endtime	subjectid	roomid	teachesid	fname	lname	staffid	subname
E1	08:40:00	09:40:00	S_01	R1	T1	Ryan	Reynolds	F1	Maths

Figure 25 - Screenshot of query

Student timetable:
This query lists all the events that a particular student is registered to, which is fundamentally the student's timetable.

```
SELECT a.*, b.roomid, c.attendsid, d.fname,d.lname, e.subject
FROM (
    SELECT eventid, starttime, endtime, dateof, subjectid FROM
    event
    ) a
    JOIN (
    SELECT roomid,eventid FROM hostedin
    ) b
    ON a.eventid = b.eventid
    JOIN (
    SELECT eventid, studentid, attendsid FROM attends
    ) c
    ON c.eventid = b.eventid
    JOIN(
    SELECT fname, lname, studentid from student where lname =
    'charge'
    ) d
    ON d.studentid = c.studentid
    JOIN(
    SELECT Subject, subjectid from subject
    ) e
    ON e.subjectid = a.subjectid
```

The results of this query are shown below.



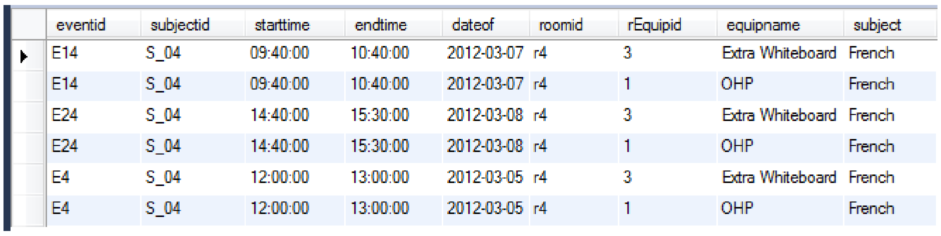
eventid	starttime	endtime	dateof	subjectid	roomid	attendsid	fname	lname	subject
E1	08:40:00	09:40:00	2012-03-05	S_01	R1	a1	Amy	Charge	Maths
E6	14:40:00	15:30:00	2012-03-05	S_06	R6	a11	Amy	Charge	Science
E20	09:40:00	10:40:00	2012-03-08	S_10	R10	a21	Amy	Charge	Technology
E15	11:00:00	12:00:00	2012-03-07	S_05	R5	a31	Amy	Charge	Spanish
E20	09:40:00	10:40:00	2012-03-08	S_10	R10	a41	Amy	Charge	Technology
E25	08:40:00	09:40:00	2012-03-09	S_05	R5	a51	Amy	Charge	Spanish
E30	14:40:00	15:30:00	2012-03-09	S_10	R10	a61	Amy	Charge	Technology

Figure 26 - Screenshot of query

Equipment Location:

The next query allows for a search of what equipment appears in which room, the search can be done either through the room id or through the name of the subject, the following query is searching with the name of the subject. This query used six tables to retrieve the details of which equipment is available for different subjects, the tables were: 'Event', 'Hostedin', 'Room', 'Equipinroom', 'Equipment' and 'Subject'.

```
SELECT a.*, c.roomid, d.rEquipid, e.equipname, f.subject FROM (
     SELECT eventid, subjectid, starttime, endtime, dateof FROM
     Event
    ) a
    INNER JOIN (
    SELECT roomid, eventid FROM Hostedin
    ) b
    ON a.eventid = b.eventid
    INNER JOIN (
    SELECT roomid FROM Room
    ) c
    ON b.roomid = c.roomid
    INNER JOIN (
    Select roomid, rEquipid FROM Equipinroom
    )d
    ON c.roomid = d.RoomId
    inner JOIN (
    SELECT EquipName, rEquipId FROM Equipment
    ) e
    ON d.rEquipId = e.rEquipId
    INNER JOIN (
     Select Subject, subjectid from Subject where subject = 'french'
    ) f
    ON f.subjectid = a.subjectid
```

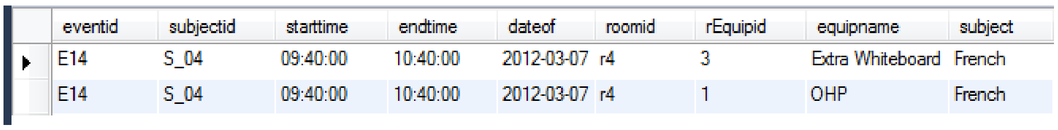


eventid	subjectid	starttime	endtime	dateof	roomid	rEquipid	equipname	subject
E14	S_04	09:40:00	10:40:00	2012-03-07	r4	3	Extra Whiteboard	French
E14	S_04	09:40:00	10:40:00	2012-03-07	r4	1	OHP	French
E24	S_04	14:40:00	15:30:00	2012-03-08	r4	3	Extra Whiteboard	French
E24	S_04	14:40:00	15:30:00	2012-03-08	r4	1	OHP	French
E4	S_04	12:00:00	13:00:00	2012-03-05	r4	3	Extra Whiteboard	French
E4	S_04	12:00:00	13:00:00	2012-03-05	r4	1	OHP	French

Figure 27 - Screenshot of query

This screenshot shows the results from the query. If we wanted to do the search for a specific time I just had to replace a part of the code. This allowed me to produce a more defined set of results specifying the time of the 'French' lesson between 8:40 and 11:00, as shown below.

```
SELECT eventid, subjectid, starttime, endtime, dateof FROM Event
WHERE starttime >= '084000'AND endtime <= '110000';
```



eventid	subjectid	starttime	endtime	dateof	roomid	rEquipid	equipname	subject
E14	S_04	09:40:00	10:40:00	2012-03-07	r4	3	Extra Whiteboard	French
E14	S_04	09:40:00	10:40:00	2012-03-07	r4	1	OHP	French

Figure 28- Screenshot of query

Code Duplication:

During the development of my database and from my research, I discovered that the creation of my link tables hadn't been done correctly. Throughout my database I created my link tables with a primary key for the table and reference my two foreign keys e.g. the 'attends' table had an 'attendsid', the two foreign keys were 'studentid' referenced from the student table and the 'eventid' reference from the event table. I later discovered that the link table should have 'studentid' and 'eventid' combined to form a composite primary key. By using this there avoids duplication across the database, without this, it is likely that there will be bugs in the code at some point. As my system deals with the planning of timetables it is imperative that my database avoids duplication.

The code below shows how my link table should be produced:

```
CREATE TABLE Attends
(
    studentId Varchar(10) NOT NULL,
    eventId Varchar(6) NOT NULL,
    CONSTRAINT PK_StudentEvent PRIMARY KEY
    (
        StudentId,
        eventId
    ),
    FOREIGN KEY (studentId) REFERENCES Student (studentId),
    FOREIGN KEY (eventId) REFERENCES Event (EventId))
```

4.4 Evaluation

The key learning's of this part of my project was primarily that I was able to learn a new subject in order to produce this part of my project. Overall I am pleased with the process of creating the database part of my project as I taught myself entirely new concepts that I hadn't encountered before. I feel that I accomplished all the tasks I set out to do, including the planning and designing of my database to a good standard. Throughout the development of creating my database I was continuously learning new material. At a later stage when I started testing my database there were some things that I noticed needed changing. Initially when creating the database I had set students to have an age and not a date of birth, this wasn't right for the needs of my project as students in the UK year groups run from September to August. This was something minor and the changing of it was quite simple.

I managed to get the database working the way that I had planned it to in my initial requirements stage. Although I encountered many problems during the process of designing and building my database I have acquired much knowledge. Due to my lack of experience with SQL I encountered more problems than I thought I would, some of these problems set me back and it meant I lost time to focus on some parts of my project. At this stage I still hadn’t looked at my user interface, and due to spending more time than I had planned on my database, it left me less time to concentrate on my user interface.

At this stage of my project I managed to achieve the following aims:
- Designing and implementing a database system to assist in the timetabling of classes in the school.
- To create a database that will deal with the scheduling of teachers to lessons.

My queries in my database are able to locate a student by their name and give the location of where they are in the school; this is through the creation of my ‘join queries’ that are shown in section 4.3.5. At this stage I hadn’t yet completed my aim of creating an information system which can locate a student, but through the use of queries this can already be accomplished in the database.

In my testing of my database I discussed that I had created my link table in the wrong manner, at that stage of my project I chose not to change the code of all my link tables. This is something that would definitely need changing, but I decided that I needed to focus on my java application. It is something that I discovered after I had designed and tested most of my database. My database does still perform the way it should but with more time to spend on the project the code duplication problems would need correcting as it could cause issues when dealing with the timetabling of the system as there will be duplicated records.

From this point on in my project the following aims have yet to be completed that will be dealt with in my java application:

- To create a student/ teacher timetabling system.
- To create an information system, where in the case of emergency someone working for the school can find out where individual students are located.
- To create a Student Information System with the ability to search the Students name and retrieve their individual timetable, and where they can be located.
- To create system that is capable of recognising clashes
- To deal with the distances of rooms and not allocate rooms to far away for a teacher or student to be able to get to in a short amount of time.

5. The User Interface

The next chapter discusses how I designed and developed my user interface.

From my research(2.2.2) I decided to use java for my application as it has relatively simple way of connecting to a database which involves writing a piece of small code that can connect the database with the program, this is automatically done in Netbeans ‘Create a database application package’.

The java development cycle can be split into four phases which are specification, design, implementation and testing.

The specification defines what the product will be able to do, this has already been decided on in my requirements analysis found in Chapter 3.

At the end of my project I wanted my system to be capable of managing the timetabling of teachers to lessons, producing individual timetables for students and also a system that is capable of dealing with locating of student and staff members for emergency reasons.

During my research I discovered that creating a system that could automatically recognise clashes was more advanced than I initially thought, it involved mathematical equations and at this stage of my project I was unable to dedicate time to focusing on this issue.

5.1 Design

For the design of my java application I used the use cases that were decided on in Chapter 3. From my use cases I developed some use case descriptions that helped me to understand how my user would react to the system and how potentially the system will react to user actions. I have given an example of a use case description below for removing a student from the database. In my appendices the use case descriptions can be found for some of my other use cases, these were created to help me with the design of my interface and java application, they can be found in Appendix B.

Use Case: Remove Student from database
Actors: User
Goal: Remove a stock student from the database
Precondition: User is already logged in
Summary: User will select and remove a student from their student list
Related use cases: Find student
Steps - main flow:

Actor action:	**System Response:**
1. Include 'Find Student	
2. Click on the student and clicks remove button	
	[Check if student exists] 3. Return confirmation check
4. Confirms the removed student	
	5. Confirms the removal and updates the student list.

Steps - exceptional flow:

Actor action:	**System Response:**
	2. [If student does not exist]. Return to step 2

The use case deals with how the system will react when a user inputs an action. In this case a user selects a student from a list to be deleted, the system checks that the student exists, if they do they are deleted from the system. If the student does not exist there is an exceptional flow which means the user is presented with an error message and returned back to step two to select a student that already exists.

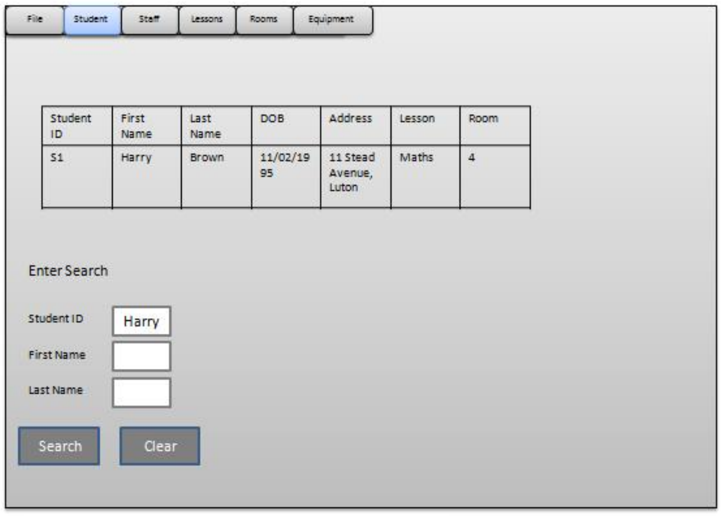
The User Interface

For the user interface I designed some prototypes of what I wanted my display screens to look like and help me decide whether they would meet my requirements. By designing the layout I avoided wasting time when actually implementing the user interface as I knew where all my buttons would be situated and I knew the functions that they would perform.

The following screen shots show what I wanted my system to look like once they were built and I have described the functions of each part of the interface.

Student → View

The following picture shows the design of my student (view) frame, it includes three text fields that the user would enter in data and a search button for the user to search, and there is also a clear button to enable the user to undo their mistakes.



Student ID	First Name	Last Name	DOB	Address	Lesson	Room
S1	Harry	Brown	11/02/1995	11 Stead Avenue, Luton	Maths	4

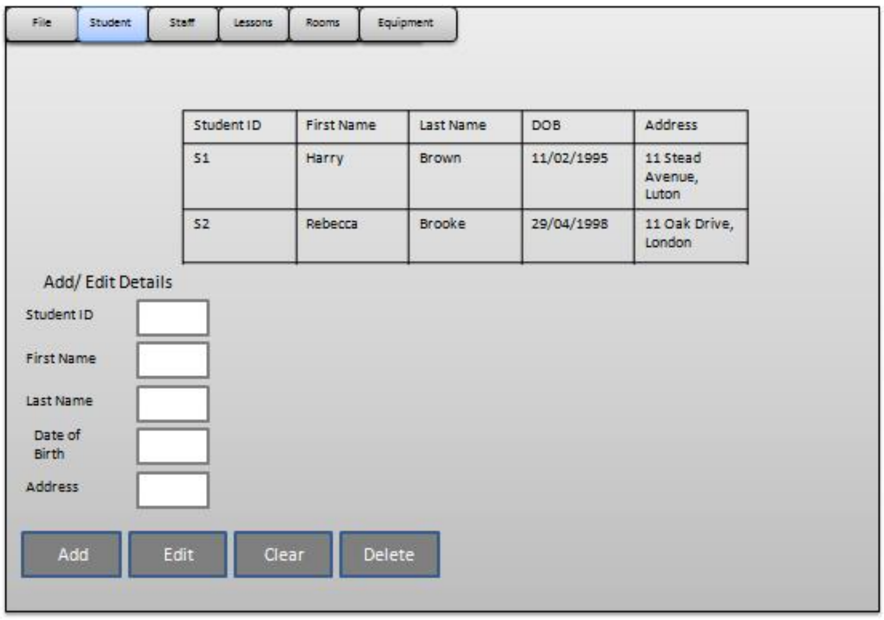
Enter Search

Student ID	Harry
First Name	
Last Name	

Figure 29 – Design – Search for student

Student → Edit

The following picture shows the design of my student (edit) frame, it includes five text fields that the user would need to enter to create a new student. If a user clicks on a student in the table they have the option of editing or deleting that student.



Student ID	First Name	Last Name	DOB	Address
S1	Harry	Brown	11/02/1995	11 Stead Avenue, Luton
S2	Rebecca	Brooke	29/04/1998	11 Oak Drive, London

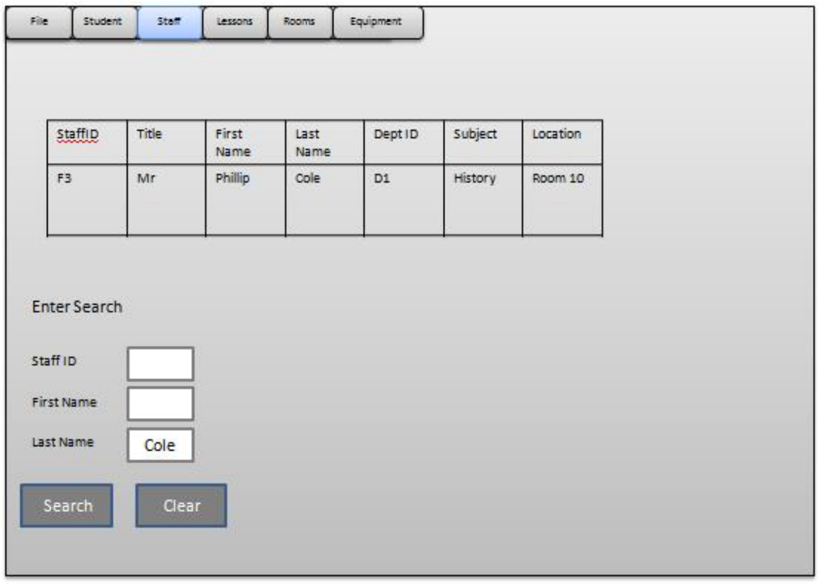
Add/ Edit Details

Student ID	
First Name	
Last Name	
Date of Birth	
Address	

Figure 30 – Design – Add/ Edit Student

Staff → View

The following picture shows the design of my staff (view) frame, it includes three text fields that the user would enter in data and a search button for the user to search for a teacher, and there is also a clear button to enable the user to delete previous entries.



StaffID	Title	First Name	Last Name	Dept ID	Subject	Location
F3	Mr	Phillip	Cole	D1	History	Room 10

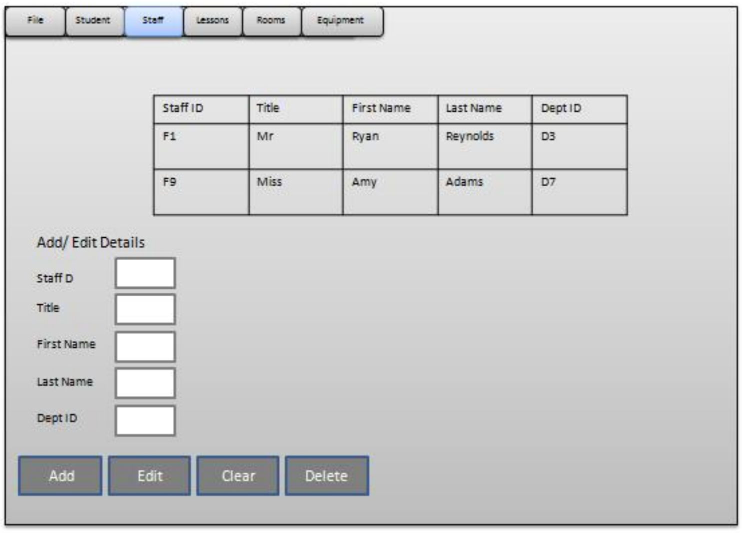
Enter Search

Staff ID	
First Name	
Last Name	Cole

Figure 31 – Design – Search for staff member

Staff→ Edit

The following picture shows the design of my staff (edit) frame, it includes five text fields that the user would need to enter to create a new staff member. If a user clicks on a staff member in the table they have the option of editing or deleting that student.



Staff ID	Title	First Name	Last Name	Dept ID
F1	Mr	Ryan	Reynolds	D3
F9	Miss	Amy	Adams	D7

Add/ Edit Details

Staff D

Title

First Name

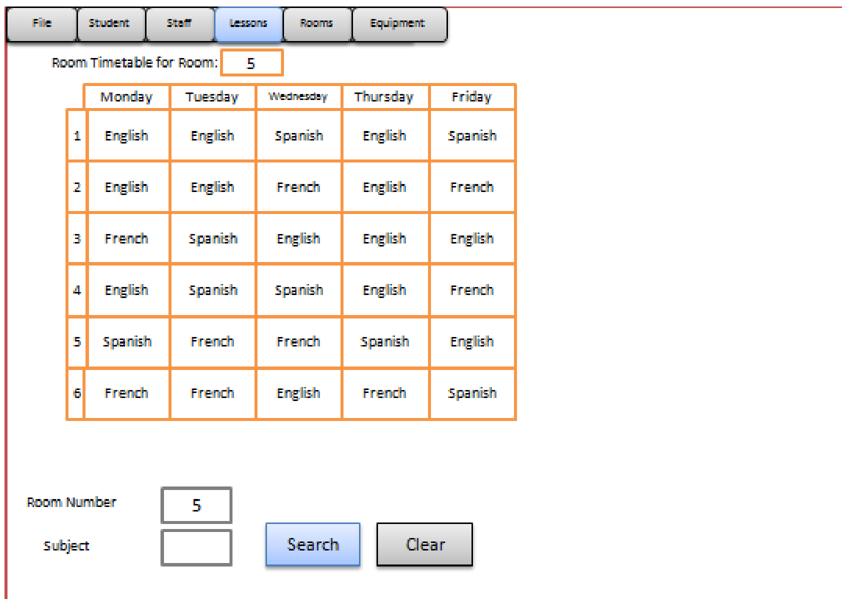
Last Name

Dept ID

Figure 32 – Design – Add/ Edit staff details

Lessons → View

The following design shows how I would like my lessons (view) screen to appear. The user can search either by subject or room number for the timetable, if the results are too large they may need to define it by subject and room number. There is a clear button for to enable the user to delete any mistakes they may have made.



Room Timetable for Room: 5

	Monday	Tuesday	Wednesday	Thursday	Friday
1	English	English	Spanish	English	Spanish
2	English	English	French	English	French
3	French	Spanish	English	English	English
4	English	Spanish	Spanish	English	French
5	Spanish	French	French	Spanish	English
6	French	French	English	French	Spanish

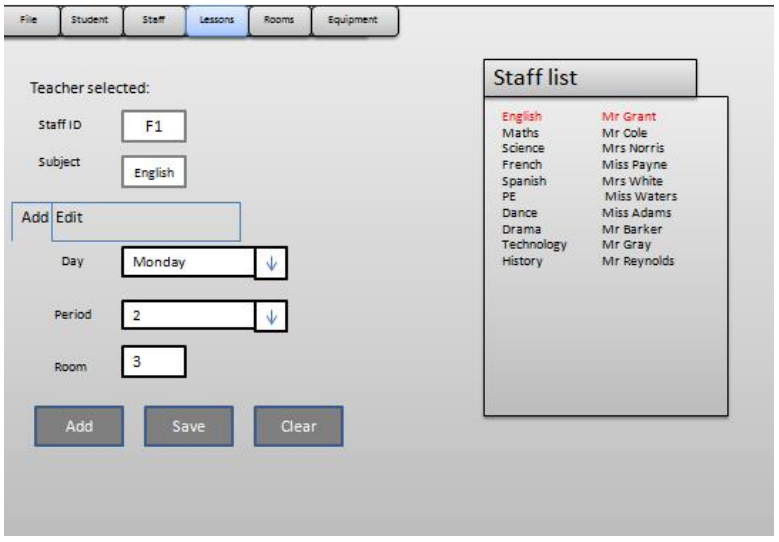
Room Number: 5

Subject:

Figure 33 – Design – View lessons (by room number or subject)

Lesson →Edit

The following picture represents what I want my edit lesson screen to look like. This part of the interface deals with the adding and editing of lessons (events). The user selects the teacher from the list on the right, once selected the details show up in the boxes in the upper left of the screen. The user can either add or edit an event by selecting the 'add' or 'edit' tab. The user can select from a drop down menu of what days are available and what periods are available. When on the 'add' tab the back end of the system will only show periods where that teacher is currently not teaching, alternatively; if they are teaching the day or period that this occurs will not show up. When on the 'edit' tab the user will be shown all events that are available and also have the option of adding an event, which before being allowed to delete they will have to confirm via a dialog box that they confirm the decision they have made.



Teacher selected:

Staff ID	F1
Subject	English
Day	Monday
Period	2
Room	3

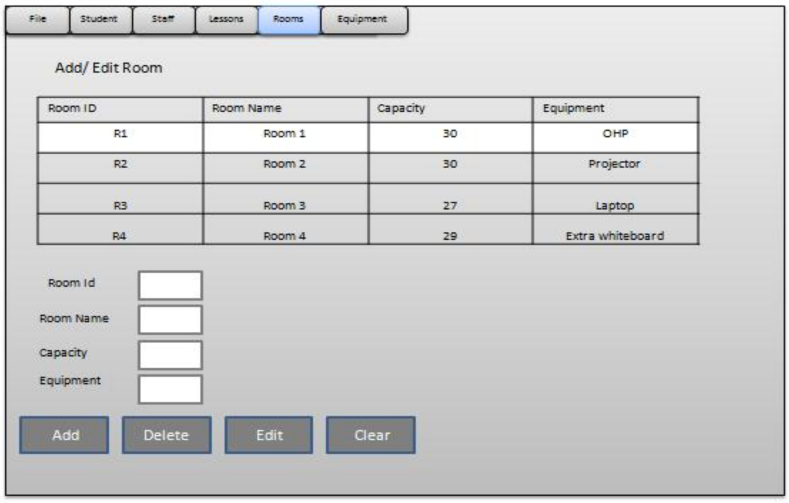
Staff list

English	Mr Grant
Maths	Mr Cole
Science	Mrs Norris
French	Miss Payne
Spanish	Mrs White
PE	Miss Waters
Dance	Miss Adams
Drama	Mr Barker
Technology	Mr Gray
History	Mr Reynolds

Figure 34 – Design – Add/ Edit lessons

Rooms → View/Edit

This design shows my room (view/edit) screen. The user is able to select from the row the room they want, they then can then make the decision to edit or delete that room. If they edit they will fill in the text boxes and select the edit button. The user can also add higher capacity or different equipment into the room. They have the option of adding a new room or clearing the text fields of data.



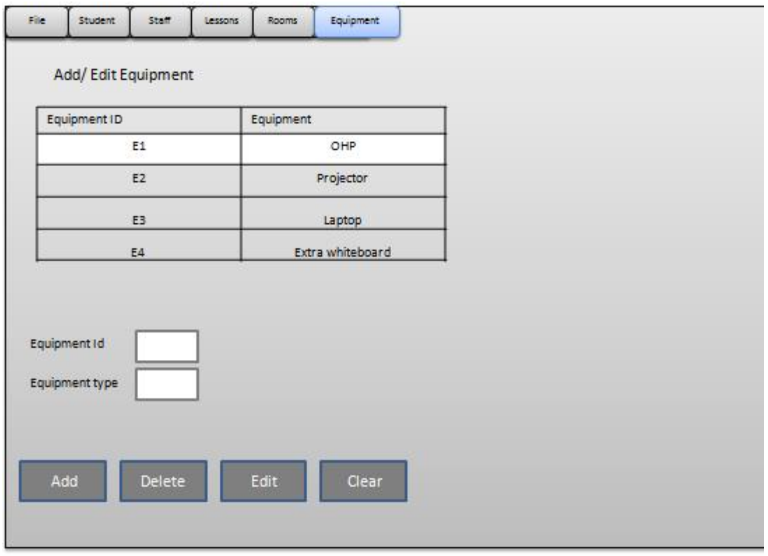
Add/ Edit Room

Room ID	Room Name	Capacity	Equipment
R1	Room 1	30	OHP
R2	Room 2	30	Projector
R3	Room 3	27	Laptop
R4	Room 4	29	Extra whiteboard

Figure 35 – Design – View/ Edit Rooms

Equipment → Add/Edit

The following design allows the user to add or remove equipment from the database.



Equipment ID	Equipment
E1	OHP
E2	Projector
E3	Laptop
E4	Extra whiteboard

Figure 36 – Design – Add/ Edit equipment

5.2 Implementation & Testing

From my research (2.2.2) using 'The Java Tutorials' (Oracle, 2012) I discovered there were many different components in java swing. The following decisions were made after I followed the tutorials found on the Oracle website, which helped me make the decisions as to what components would be right for my application.

After researching what each component was capable of I decided which would be most suitable for my application and the reasons why I chose them are discussed below.

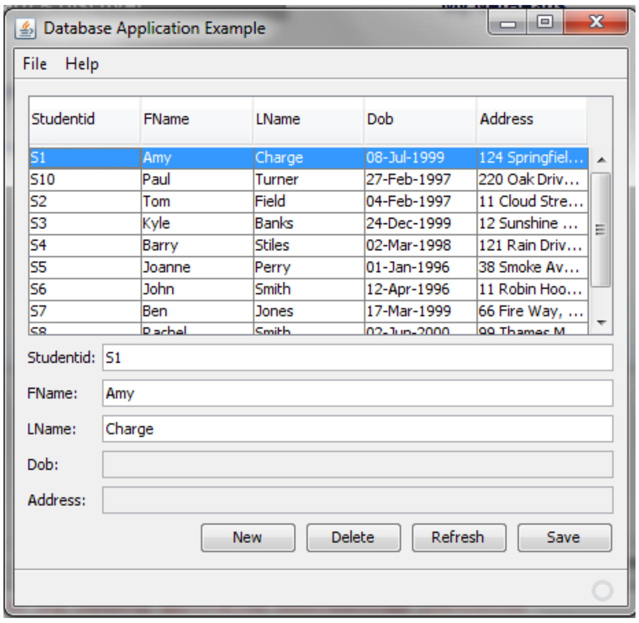
JDialog is a top level container which is a component that can contain other components; a container can only be contained once. It also enables the use of a content pane, which will allow me to add other panels to the JDialog. A top level container also allows the addition of a menu bar, which is necessary for the type of application I wanted to make. I chose to use JDialog as I wanted a non-modal dialog to come up if there were any problems, this would mean if any error happened, by having a non-modal dialog the user is not forced into doing anything. If I were to use a modal dialog it would generally be seen as bad designing of usability.

JPanel, JScrollPane, JButton and JTable all inherit from the JComponent class, throughout the design of my interface these components will be an important part of my application.

Within Swing there is a layout manager which helps when creating the layout of the GUI. The layout manager that I decided to use was CardLayout, this is because it allows panels to be stacked upon each other, with only one panel visible at a time. As my interface is going to have many different panels holding different pieces of information, I felt this was a good choice as I have many different parts to my interface and I could switch panels with the click of a button.

Initially when starting this project I had an idea of how I was going to create my java application. From the tutorials I followed online it seemed like this process wouldn't be too hard. But when it actually came to doing it, I encountered many problems.

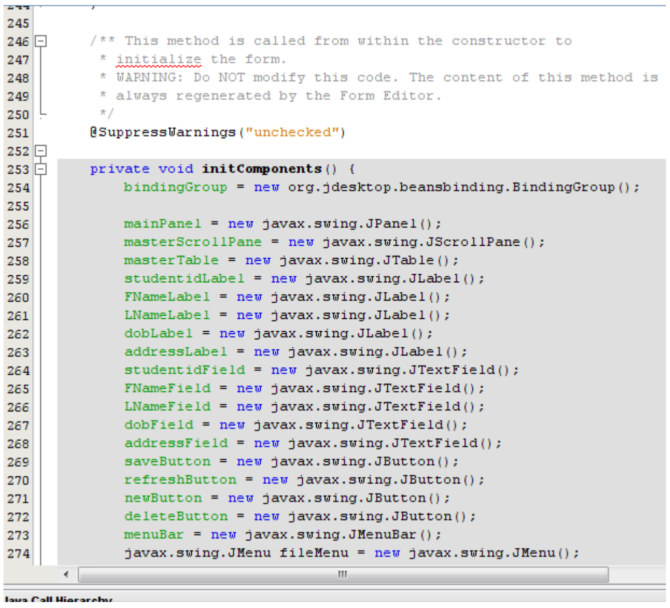
The first version of my project used the Netbeans IDE 'Create Desktop Application' function, initially when starting this project I thought I would be able to adapt the code created by Netbeans and change the application to suit the needs of my design. When using this method there is only the option of selecting one table to connect to, at the time I thought I would then be able to add in the other tables, but unfortunately this didn't work. The screen shot below gives an example of what can actually be created, this application was automatically generated by the Netbeans IDE.



Studentid	FName	LName	Dob	Address
S1	Amy	Charge	08-Jul-1999	124 Springfiel...
S10	Paul	Turner	27-Feb-1997	220 Oak Driv...
S2	Tom	Field	04-Feb-1997	11 Cloud Stre...
S3	Kyle	Banks	24-Dec-1999	12 Sunshine ...
S4	Barry	Stiles	02-Mar-1998	121 Rain Driv...
S5	Joanne	Perry	01-Jan-1996	38 Smoke Av...
S6	John	Smith	12-Apr-1996	11 Robin Hoo...
S7	Ben	Jones	17-Mar-1999	66 Fire Way, ...

Studentid: S1
FName: Amy
LName: Charge
Dob:
Address:

Figure 37 – Example of Netbeans IDE creation



```
    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")

    private void initComponents() {
        bindingGroup = new org.jdesktop.beansbinding.BindingGroup();

        mainPanel = new javax.swing.JPanel();
        masterScrollPane = new javax.swing.JScrollPane();
        masterTable = new javax.swing.JTable();
        studentidLabel = new javax.swing.JLabel();
        FNameLabel = new javax.swing.JLabel();
        LNameLabel = new javax.swing.JLabel();
        dobLabel = new javax.swing.JLabel();
        addressLabel = new javax.swing.JLabel();
        studentidField = new javax.swing.JTextField();
        FNameField = new javax.swing.JTextField();
        LNameField = new javax.swing.JTextField();
        dobField = new javax.swing.JTextField();
        addressField = new javax.swing.JTextField();
        saveButton = new javax.swing.JButton();
        refreshButton = new javax.swing.JButton();
        newButton = new javax.swing.JButton();
        deleteButton = new javax.swing.JButton();
        menuBar = new javax.swing.JMenuBar();
        javax.swing.JMenu fileMenu = new javax.swing.JMenu();
```

Figure 38 – Netbeans IDE auto-generated code

When I tried to adapt the code to make my own, it was hard to edit what the actual IDE had made. The picture above (fig. 38) shows the initComponents() method, which is the method that holds all of the Swing components functionality, highlighted in grey is the section which couldn't be edited. This meant I had to change my approach to creating my application as I was unable to add my own functionality to the interface that Netbeans generated. The below gives an example of what happened when I tried to add my own functionality. It seemed to cause problems for the table as the data originally displayed and the text fields disappeared.

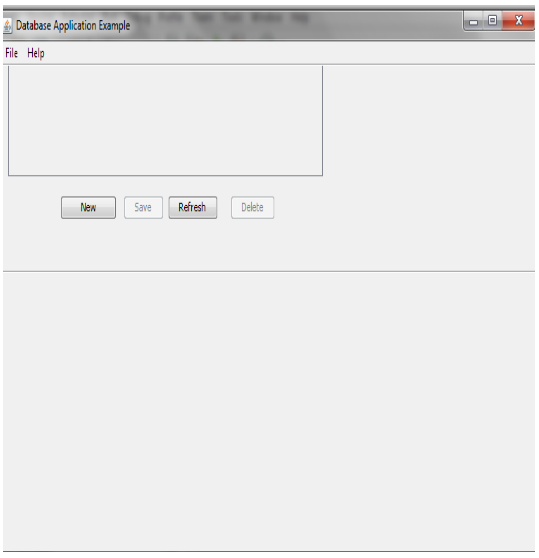


Figure 39 – Netbeans Database Application after editing

The Java Persistence API

From this point onwards I tried to look into different ways of making my application. From my research (2.2.2) I was able to find a new method that would be more appropriate for my purpose which was the Java Persistence API. After following tutorials of how I could do this I was ready to try creating my own.

The following section discusses how I went about creating my user interface.

Within Netbeans IDE I created a Java Class Library. I created a GUI package to contain my user interface class. I then created a School package which I used to test my methods were working before implementing them in the user interface to check they worked correctly. The 'DBManage' package contains my entity classes, which were automatically generated by the Netbeans IDE. By separating my classes into different packages it enabled me to organise and understand my classes, if they were all mixed up it would be hard for me to understand what each part did.

Entity Class

As discussed in my research (2.2.2) an entity class is used to represent objects in a database. The following piece of code is found in my 'DBManage' package, this piece of code is automatically generated by Netbeans, I will discuss what it does as I was able to create it myself, but as it is capable of being automatically generated by Netbeans, it was easier to do this for my twelve tables, rather than spend time doing it myself. Within Netbeans there is an option to add entity classes from databases, from this you connect to the appropriate database, which then auto generates the code for each table, including queries and get methods and set methods. I have selected a small sample of the code from the Student entity, the rest of the entity class can be found in the appendices.

```
@Entity
@Table(name = "student", catalog = "project", schema = "")
@XmlRootElement
@NamedQueries({
    @NamedQuery(name = "Student.findByAddress", query =
"SELECT s FROM Student s WHERE s.address = :address")})
public class Student implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @Basic(optional = false)
    @Column(name = "studentid", nullable = false, length = 6)
    private String studentid;
```

In the constructer, the student attributes are declared and are created exactly how they are represented in my database. The 'studentid' which was Varchar in my database is now referred to as a String in the java code, the Netbean IDE automatically converts it to the value it sees as a best fit. @Entity signifies that this class is an entity class. @ID means it is a primary key. @Column refers to the attributes in the tables of my database. The entity class also includes auto generated queries that are useful such as 'student.findbyAddress', this can help me later on when creating methods for searching for a student via their address.

The User Interface

In this class I designed my interface and it is also where I managed to implement some of my methods. The method that I managed to fully get working was adding a student to a database and I managed to partially get working my edit a student. The reason why this isn't fully functional is because although it managed to edit an existing student, if you try editing a student that didn't exist then it would add that student to the database, creating a new student which isn't what should happen. In an ideal situation an error would have been returned saying 'This student doesn't exist' but I didn't manage to do this.

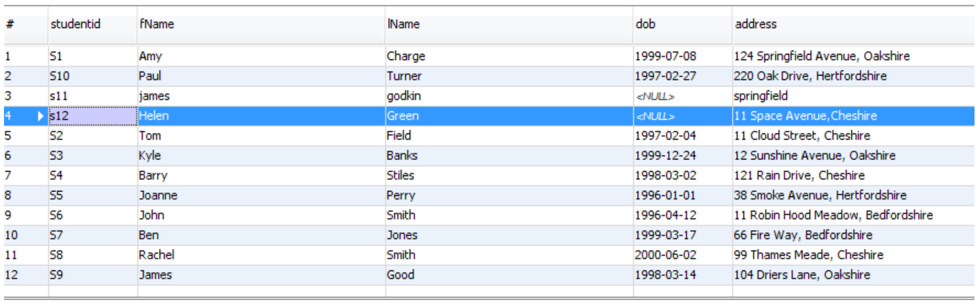
To test that my create method worked I created a class called School, which was my test class. The following piece of code creates a new student, it sets 'studentid', 'fname', 'lname' and 'address' to the values indicated below

```
   private static void m1() throws Exception  {

     Student student = new Student();
     student.setStudentid("s12");
     student.setFName("Helen");
     student.setLName("Green");
     student.setAddress("11 Space Avenue,Cheshire ");
     System.out.println();


     myEntityManager.getStudentJpaController().create(student
);}
```

To check the method worked I did a query select * from student which can be shown below, it shows the method was successful as the student ‘Helen’ can be seen in the screen shot below in row 4.



#	studentid	fName	lName	dob	address
1	S1	Amy	Charge	1999-07-08	124 Springfield Avenue, Oakshire
2	S10	Paul	Turner	1997-02-27	220 Oak Drive, Hertfordshire
3	s11	james	godkin	<NULL>	springfield
4	s12	Helen	Green	<NULL>	11 Space Avenue,Cheshire
5	S2	Tom	Field	1997-02-04	11 Cloud Street, Cheshire
6	S3	Kyle	Banks	1999-12-24	12 Sunshine Avenue, Oakshire
7	S4	Barry	Stiles	1998-03-02	121 Rain Drive, Cheshire
8	S5	Joanne	Perry	1996-01-01	38 Smoke Avenue, Hertfordshire
9	S6	John	Smith	1996-04-12	11 Robin Hood Meadow, Bedfordshire
10	S7	Ben	Jones	1999-03-17	66 Fire Way, Bedfordshire
11	S8	Rachel	Smith	2000-06-02	99 Thames Meade, Cheshire
12	S9	James	Good	1998-03-14	104 Driers Lane, Oakshire

Figure 40 – Screenshot of query

The method create() can be found in the package Ctrl which is where my StudentJPAController class was located.

JPA Controller:
The StudentJPAController class is generated by Netbeans IDE; it is where the entity manager factory pattern that is used to make an Entity Manager is located (Rose India Technologies, 2008). The entity manager is responsible for maintaining the active collection of entity objects that are being used by the application. The Entity Manager handles the database interaction and meta-data for object-relational mappings. The entity objects refer to the class that represents the rows in the database.

Update Table
The following method deals with updating the table located in the user interface, which holds details of students found in the database.
It retrieves the students from the database and for every student, a student object is created, it is then added to the rowData(which is the data in the table) which is then added to the DefaultTabelModel dtm. When adding a student this method will automatically add the student to the list.

```
private void fillDate() {
DefaultTableModel dtm =(DefaultTableModel) studtbl.getModel();
dtm.setRowCount(0);
List<Student> students =
myEntityManager.getStudentJpaController().findStudentEntities(
);
for(Iterator<Student> it = students.iterator();
it.hasNext();){
Student student = it.next();

Object [] rowData ={student.getStudentid(),student.getFName()
,student.getLName(),student.getDob(),student.getAddress()};
dtm.addRow(rowData);
}}
```

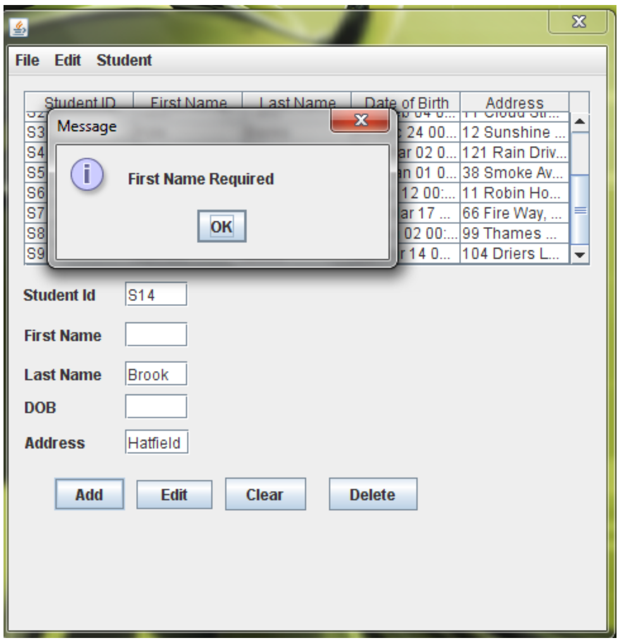
Add Student

I then created a method 'add()' which would add a student from details that had been typed into a text field. The code is triggered by the pressing of the add button on my user interface. A new student is created, in each text field the value is read and is set to what the user typed, there is also a check for whether the text fields are empty and if they are it will return a message dialog telling them to fill in the missing value. The create method is then called from the JPA controller, then filldate() is executed and adds the newly created student to the table. The code is displayed below.

```
private void Add() throws PreexistingEntityException,
Exception {
Student student = new Student();
student.setStudentid(txtid.getText().trim());
student.setFName(txtfname.getText().trim());
student.setLName(txtlname.getText().trim()) ;
student.setAddress(txtaddress.getText().trim());
String studentid = txtid.getText().trim();
String fname = txtfname.getText().trim();
String lname = txtlname.getText().trim();
String address = txtid.getText().trim();
// student.setDob(new Date() = (Date)txtdob.getValue());

MyEntityManager.getStudentJpaController().create(student);
clear();
fillDate();
}}
```

Error Message



First Name Required

Student Id S14

First Name

Last Name Brook

DOB

Address Hatfield

Figure 41 – Screenshot of application

The screenshot above shows what happens when the add button is selected and the first name text box is left empty. It returns a message dialog saying that the first name is required. The following code shows how I manage to do this. If the 'First Name' textbox is empty, the 'Joptionpane' is called showing the message "First Name Required".

```
if(fname.equals("")){
JOptionPane.showMessageDialog(this, "First Name Required");
txtfname.requestFocus();
return;
}
```

Edit Student
The following code is how I created my edit student method, at this time it allows the user to edit a student that exists, but if it edits a student that doesn't exist it creates a new student which is something that needs to be further queried.

```
private void EditStudent() throws NonexistentEntityException {
    Student student = new Student();
    student.setStudentid(txtid.getText().trim());
    student.setFName(txtfname.getText().trim());
    student.setLName(txtlname.getText().trim()) ;
    student.setAddress(txtaddress.getText().trim());
    try {

myEntityManager.getStudentJpaController().edit(student);
    } catch (Exception ex) {

Logger.getLogger(View.class.getName()).log(Level.SEVERE, null,
ex);
    }
   clear();
   fillDate();

}}
```

The full code my my project can be found in Appendix H
Testing:
There are many different ways of testing a java system, as I had quite a few issues with my coding not working. I found it was in my best interests to test each method as I created and modified it. By using the Netbeans IDE debugger I was able to locate where some of my problems were located. The Netbeans IDE debugger allows you to run your system line by line. Breakpoints are added to the line number and by selecting the 'Debug Project' Icon, it allowed me to control what part of the application ran and I could step by step find where the source of the problem came from. Although Netbeans IDE is capable of giving error messages, the code I had was located in three of my classes which meant that the problem was coming from one of three places. By using this method I managed to successfully find where the problem was located.

5.3 Evaluation

As I had experience of using java from course modules I had taken, I think my approach to starting my java application little lightly. I encountered more problems than I thought I would and it generally came down to the misconception of thinking I knew enough about java already to complete this part of my project. After dealing with a few problems with my database, I actually started the design and implementation of the java application later than I expected. By starting later I wasn't able to fully deal with many of the problems that I encountered on the java side of my project and some coding issues that I had took hours to resolve as it is not always clear where the problem is.

From my research I was able to discover that producing a system that is able to deal with recognising timetable clashes was actually more advanced than I thought, it involved creating equations that could solve the problem, and at this stage the problem hasn't been fully solved (Sandhu, 2003). This is an aim of my project that I couldn't achieve due to it being a more advanced topic than I initially thought it would be.

The first problem that I had to overcome with the development of my application was initially using the 'create a java database application' within the Netbeans IDE. I hoped to be able to create the application using Netbeans IDE auto generated method. From my experience I found that I was unable to edit the code without experiencing further problems. This meant I had to find a different method for creating my application.

From my research I discovered that the Java Persistence API was the most appropriate for my needs. I managed to get a part of my application working which was being able to add and edit a student in the database. The reason why I only focused on the student part of my application was due to the fact that I was having many issues with the implementation of my application due to numerous coding problems. I decided that if I focused on a small part of my application I would be able to achieve more than I would have if tried to focus on the more advanced parts of my application. I thought it would be more beneficial if I tried completing a small part of the system, rather than having a system that couldn't do anything.

I also wasn't able to achieve my aim of dealing with room distance when timetabling, this is something I planned to develop after the creation of the java application had been completed, unfortunately due to time constraints it was an aim that couldn't be looked at considering the main part of my system wasn't fully functional.

Although I didn't fulfil my aims with my java application I was able to find a potential solution for creating the system in the future. As I ran out of time I was unable to spend time working out the coding problems I was having with my java system, with more time to spend on the system I could potentially find a solution and eventually create the working system I had planned in the aims of my project. Using java persistence with more experience in java is definitely the right method to use when connecting the database to a java application; I needed more time to understand the subject matter before fully implementing my system.

6. Conclusion

My technical conclusion discusses what my aims and objectives were and whether I achieved them. It also discusses what could be done differently with the project.

My personal conclusion discusses the skills I developed, my time management and if there was anything I could do differently if this project was done again.

6.1 Technical Conclusion

In my introduction (1.1) I listed the aims and objectives that I wanted to achieve to enable me create the system.

I was able to achieve my aim of creating a database that was fully able to deal with the scheduling of teachers and students to lessons. Although I did have some issues in creating my database I did manage to overcome most of the problems I had.

The original aim of my project was to produce a working system that was capable of dealing with the timetabling of students and teachers. Although didn't fully achieve this aim, I was able to create a database that was fully functioning but the java application I managed to produce would not be capable of producing timetables for students or teachers. When trying to find resources that could help me with the problems I was having, I struggled to find the appropriate information. Most of the support information I found for using the Java Persistence API was written for producing a web application which meant that it wasn't relevant for my project. Also when researching applications for use with databases, I found that most of the resources available were directed towards using PHP and at the beginning of my project I was adamant that I was going to use java due to my previous experience of using it.

My database is able to carry out the functions of my system and I was able to produce advanced queries which allowed me to deal with the time tabling of students and teachers.

At the start of my project I created a list of objectives to help me achieve my project aims. Overall I managed to complete most of these objectives. The objectives that weren't completed were to do with my java application, as I had problems with the development of it. Another objective that wasn't completed was creating a system that dealt automatically with clashes, from my research I determined that this was not something I would be able to achieve in the time frame I had as it involves numerous mathematical equations and I would have been a new concept I needed to learn.

To further develop this project, I would look into how dealing with timetable clashes could be solved and with the resources available I would create a web application in PHP rather than java. The amount of facilities online to assist database developers in creating database applications is abundant and I believe with those resources I would be able to create the system that I wanted to meet the aims of my project.

6.2 Personal Conclusion

Initially when starting this project I had basic knowledge of databases and I had never designed or created one. From this project I have learnt a complete new subject and now have good confidence in my ability to create one again. From my experience I have enjoyed learning this skill and it has changed my ideas for what I want to do when I leave university. Prior to starting this project I wouldn't have considered a career within database management, but I am now actively looking for roles in it.

Generally at the start of this project I felt that the java application was going to be the easier part to complete and that the database would be the hardest part. In fact it wasn't as I thought at all. When it came to the java side of my project, I came across more problems than I expected and felt that there were more resources available to web developers for use with java persistence. Unfortunately the misconception that I knew enough about java initially is a reason why I couldn't complete the project. Although I had experience with java applications, I had never made an application that could interact with a database and this was a harder task than I thought it would be. As I started dealing with this problem late into my timescale it was unfortunate that I was unable to achieve my aims of creating a system capable of timetabling students and teachers. It was frustrating not being able to complete a project that I had spent a lot of time on, but I understand in reality that the timescale set was not enough time for me to achieve all my aims.

I enjoyed learning how to create databases, how they functioned and what can be achieved using them. The process of learning how to create and manipulate databases that I followed in the research of this project was extremely important to helping me develop a fully functional database that met all my user requirements. For some problems that I did have with the databases I was able to receive help from my tutor which was extremely helpful, without this support I may not have created a fully working database system. Overall I am pleased with the development of my database and as I knew little about creating a database before starting this project and feel that I accomplished quite a large achievement.

Throughout my project I had to deal with time management and overall I think I managed well, this was mainly because I focused the majority of my attention on my project and not my other modules. At the start of the project I found it hard to focus as I was concentrating on too many parts of the project at once. In the Gantt chart appendix A, I didn't allow time for my requirements analysis which is extremely important in a project. Once I had fitted requirements into the time scale of my project changed and I didn't follow the Gantt chart closely as I felt it was created at a time when I didn't understand my project fully. Looking back on what I've achieved I am happy with what I managed to accomplish, although I would have liked to have had the system working fully. If I could change anything about my time management it would be to stick to a set of tasks as I found I was spreading myself too thinly trying to focus on everything.

By spending more time on the creation and testing of my database this lead to my java application not being completed. If I were to be able to repeat the project again, I wouldn't change the amount of time spent producing my database system. Capturing the ideal requirements and completing the database to a high standard was an extremely important part of my project. The java application could only be built once the database could meet the user requirements that were decided upon in my requirements analysis.

If I could do this project again I would do much the same, but I would try to create my timetabling system as a web application in PHP, due to the amount of resources available to help database developers.

Overall I have enjoyed the experience of creating and developing my own project. Throughout my time at university I have completed a number of group projects and projects for certain modules, but I hadn't had the chance to manage a project of my own creation until this final year. I feel that the skills I have developed will be beneficial in my future career. I have managed to improve my project management skills and without improving them I wouldn't have managed to achieve most of my objectives.

Although writing this report and creating my system has been a challenge it has also been an important learning experience where my knowledge on databases has greatly improved. I have enjoyed participating in a project where I decided my learning outcomes and was able to develop new skills.

Works Cited

Ambler, S., 2010. *Database Testing: How to Regression Test a Relational Database.* [Online]
Available at: http://www.agiledata.org/essays/databaseTesting.html
[Accessed 28 03 2012].

Basics of Computer, 2012. *Definition of Database Management System (DBMS) and Components of Database Management System Environment.* [Online]
Available at:
http://www.basicsofcomputer.com/definition_of_database_management_system_and_components_of_database_environment.htm
[Accessed 14 02 2012].

Connolly, T. a. B. C., 2002. *Database Systems, A practical approach to Design, Implementation and Management.* London: Pearson Education Ltd.

Das, A., 2008. *Understanding JPA, Part 1: The object-oriented paradigm of data persistence.* [Online]
Available at: http://www.javaworld.com/javaworld/jw-01-2008/jw-01-jpa1.html?page=3
[Accessed 20 03 2012].

Das, A., 2009. *The object-oriented paradigm of data persistence.* [Online]
Available at:
http://www.yin.ch/minSpace/myDominoCorner.nsf/vwSearch/FF2B7EA1C6CA2513C12575D3002AEF35?Open
[Accessed 20 03 2012].

Database Dev, 2003. *Relational Database Design.* [Online]
Available at: http://www.databasedev.co.uk/feedback.html
[Accessed 28 02 2012].

Evans, G., 2004. *IBM : Developer Works , Getting from use cases to code, Part 1: Use-Case Analysis.* [Online]
Available at: http://www.ibm.com/developerworks/rational/library/5383.html
[Accessed 04 02 2012].

IBM, 2011. *Values for Key Attributes.* [Online]
Available at:
http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/index.jsp?topic=%2Fcom.ibm.db2z10.doc.intro%2Fsrc%2Ftpc%2Fdb2z_valuesforkeyattributes.htm
[Accessed 28 02 2012].

Jewett, T., 2006. *Discussion: more about domains.* [Online]
Available at: http://www.tomjewett.com/dbdesign/dbdesign.php?page=domains.php
[Accessed 03 25 2012].

Lim, E., Srivastava, J., PrabhakarS & Richardson, J., 1993. Entity Identification in Database Integration. *Data Engineering, 1993. Proceedings. Ninth International Conference,* 1(1), pp. 294-301.

Maguire, M. ,. B. N., 2002. User requirements analysis - A Review of supporting methods. *Proceedings of IFIP 17th World Computer Congress,* 1(1), pp. p133-148.

Malan, R. &. B. D., 2001. *Defining Non-Functional Requirements.* [Online]
Available at: http://www.bredemeyer.com/pdf_files/NonFunctReq.PDF
[Accessed 20 March 2012].

Microsoft, 2005. *Planning Phase.* [Online]
Available at: http://technet.microsoft.com/en-us/library/bb497062.aspx
[Accessed 23 03 2012].

MySQL, 1997-2012. *FOREIGN KEY Constraints.* [Online]
Available at: http://dev.mysql.com/doc/refman/5.5/en/innodb-foreign-key-constraints.html
[Accessed 27 3 2012].

ObjectDB, 2010. *Fast Object Database for Java - with JPA/JDO support.* [Online]
Available at: http://www.objectdb.com/java/jpa/getting/started
[Accessed 28 03 2012].

Oracle® Database , 2008. *2 Day Developers Guide.* [Online]
Available at:
http://docs.oracle.com/cd/B28359_01/appdev.111/b28843/tdddg_creating.htm
[Accessed 20 02 2012].

Oracle, 1995-2012. *Establishing a Connection.* [Online]
Available at: http://docs.oracle.com/javase/tutorial/jdbc/basics/connecting.html
[Accessed 30 3 2012].

Oracle, 2012. *The Java Tutorials: Using Swing Components.* [Online]
Available at: http://docs.oracle.com/javase/tutorial/uiswing/components/index.html
[Accessed 14 3 2012].

Rational Software Corporation, 2002. *Business Entity.* [Online]
Available at:
http://sce.uhcl.edu/helm/rationalunifiedprocess/process/modguide/md_bent.htm
[Accessed 17 02 2012].

Rose India Technologies, 2008. *Java Persistence API.* [Online]
Available at: http://www.roseindia.net/ejb/JavaPersistenceAPI.shtml
[Accessed 1 04 2012].

Sandhu, K. S., 2003. Automating Class Schedule Generation in the Context of a University Timetabling Information System. *Griffith University,* 1(1).

Venkatasamy, J., 2012. *JPA for Simplified Persistence: An Introduction to the Java Persistence API |*. [Online]
Available at: http://javaboutique.internet.com/tutorials/jpa-intro/index-3.html
[Accessed 29 03 2012].

Bibliography

Beginner SQL. 2012. *SQL Integrity Constraints*. [ONLINE] Available at: http://www.beginner-sql-tutorial.com/sql-integrity-constraints.htm. [Accessed 01 March 2012].

Bob Bryla, 2004. *Oracle Database Foundations: Technology Fundamentals for IT Success*. 1 Edition. Alameda**.** Sybex.

Brown, K. 2012. *MySQL - Advanced Queries*. [ONLINE] Available at: http://www.keithjbrown.co.uk/vworks/mysql/mysql_p8.php. [Accessed 18 March 2012]

Demo Source and Support. 2009. *Adding a constraint to an existing table: add foreign key to table*. [ONLINE] Available at:http://www.java2s.com/Code/PostgreSQL/Constraints/Addingaconstrainttoanexistingtable addforeignkeytotable.htm. [Accessed 29 February 12]

DuBois, P. 2005. MySQL (3rd Edition). 3 Edition. Indiana. Sams.

IBM.2012. *Foreign key (referential) constraints*. [ONLINE] Available at: http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/index.jsp?topic=%2Fcom.ibm.db2.luw.admin.dbobj.doc%2Fdoc%2Fc0020153.html. [Accessed 18 April 2012].

Mark Priestley, 2007. *Practical Object-Oriented Design with UML*. 2nd Edition. Maidenhead. McGraw-Hill.

Meersman, R. 2000. *Databases, Information and Conceptual Schema Design*. [ONLINE] Available at:http://www.starlab.vub.ac.be/staff/robert/Information%20Systems/Halpin%203rd%20ed/Infosys%20Ch1.pdf. [Accessed 13 March 12].

Mozoran. (2011). *Java & MySQL - Basic connection and insert values (NetBeans)*. [Online Video]. 13 May. Available from: http://www.youtube.com/watch?v=bO7O9tS8qe4. [Accessed: 29 February 2012].

MySQL .2012. MySQL *5.0 Reference Manual : 8.3.1.7 LEFT JOIN and RIGHT JOIN Optimization*. [ONLINE] Available at:http://dev.mysql.com/doc/refman/5.0/en/left-join-optimization.html. [Accessed 18 March 2012].

Netbeans. 2011. *NetBeans Platform CRUD Application Tutorial*. [ONLINE] Available at:http://platform.netbeans.org/tutorials/nbm-crud.html. [Accessed 21 March 12].

Netbeans. 2011. Using Hibernate with the Java Persistence API and JSF 1.2. *[ONLINE]* Available at:http://netbeans.org/kb/68/web/hibernate-jpa.html. [Accessed 12 March 12].

Nyffenegger,R. 2012. *Integrity Constraints [Oracle]*. [ONLINE] Available at: http://www.adp-gmbh.ch/ora/misc/integrity_constraints.html. [Accessed 18 April 2012].

Oracle. 2005. *Datatypes*. [ONLINE] Available at:http://docs.oracle.com/cd/B19306_01/server.102/b14200/sql_elements001.htm#SQLRF0021. [Accessed 21 February 12]

SQLite. 2012. *SQLite Foreign Key Support*. [ONLINE] Available at: http://sqlite.org/foreignkeys.html. [Accessed 18 April 2012].

The Java Tutorials. 2011. *How to Use Tables*. [ONLINE] Available at:http://docs.oracle.com/javase/tutorial/uiswing/components/table.html#fire. [Accessed 14 February 12].
Tizag. 2012. MySQL Tutorial - Time. *[ONLINE] Available at:http://www.tizag.com/mysqlTutorial/mysql-time.php. [Accessed 8 March2012].*

W3Schools. 2012. *SQL FOREIGN KEY Constraint*. [ONLINE] Available at: http://www.w3schools.com/sql/sql_foreignkey.asp. [Accessed 10 2012].

Wang, R; Kon, H; Madnick,S. 1992. Data Quality Requirements Analysis and Modeling. *Ninth International Conference of Data Engineering*, [Online]. 1, 1-14. Available at: http://web.mit.edu/tdqm/www/tdqmpub/IEEEDEApr93.pdf[Accessed 20 March 2012].

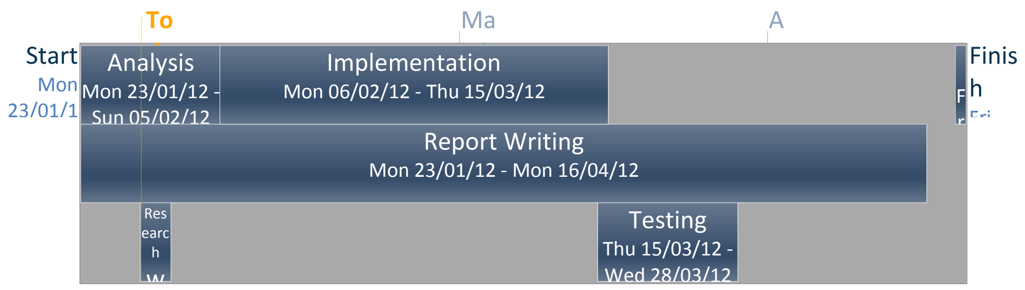
Wantededu. *(2010). Java Persistence (JPA) with MySql Database part1.avi*. [Online Video]. 30 December. Available from: http://www.youtube.com/watch?v=PFUs2Cs31jo&feature=autoplay&list=UUDnjAgLd1XnzWVtfwf9_6UQ&lf=plcp&playnext=1. [Accessed: 21 March 2012]

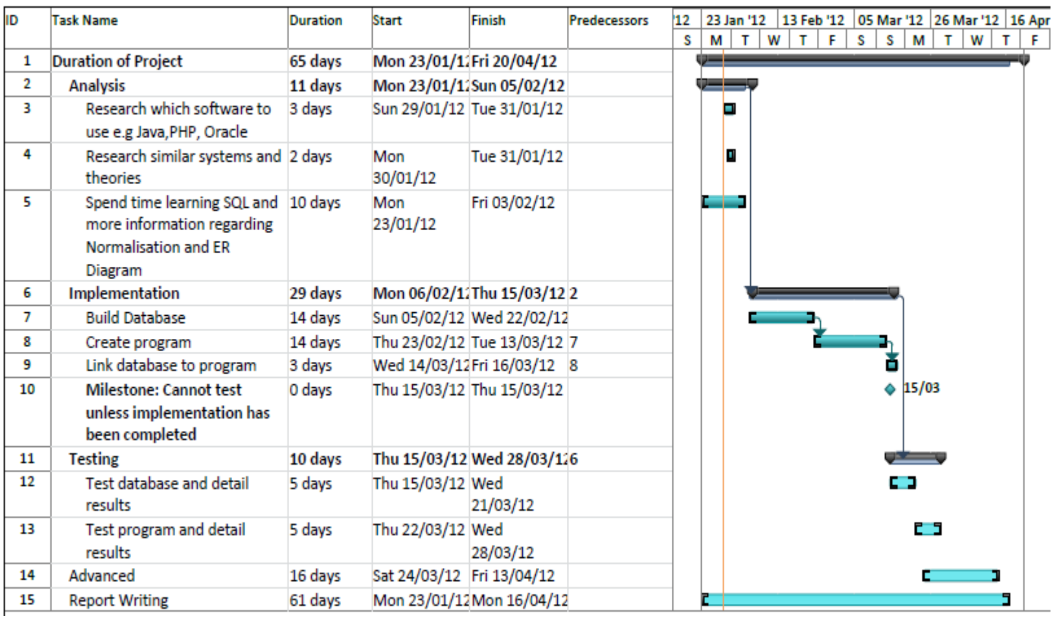
Wood M. 2012. *The Java Persistence API.* [ONLINE] Available at:http://www.studynet1.herts.ac.uk/crs/11/6COM0276-0206.nsf/Teaching+Documents/80257949007C1614802576C4007110D3/$FILE/20%20Persistence-API-V2.pdf. [Accessed 02 April 2012].

Appendices

Appendix A

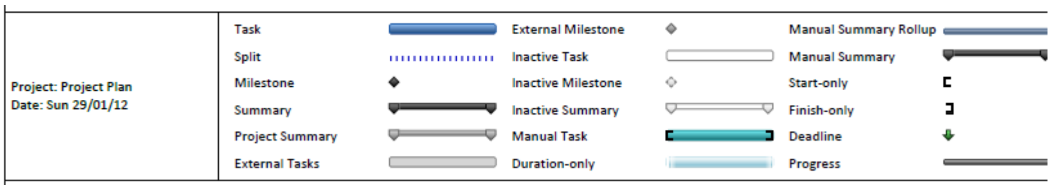
Gantt chart:





ID	Task Name	Duration	Start	Finish	Predecessors
1	**Duration of Project**	**65 days**	**Mon 23/01/1**	**Fri 20/04/12**	
2	**Analysis**	**11 days**	**Mon 23/01/1**	**Sun 05/02/12**	
3	Research which software to use e.g Java,PHP, Oracle	3 days	Sun 29/01/12	Tue 31/01/12	
4	Research similar systems and theories	2 days	Mon 30/01/12	Tue 31/01/12	
5	Spend time learning SQL and more information regarding Normalisation and ER Diagram	10 days	Mon 23/01/12	Fri 03/02/12	
6	**Implementation**	**29 days**	**Mon 06/02/1**	**Thu 15/03/12**	**2**
7	Build Database	14 days	Sun 05/02/12	Wed 22/02/12	
8	Create program	14 days	Thu 23/02/12	Tue 13/03/12	7
9	Link database to program	3 days	Wed 14/03/12	Fri 16/03/12	8
10	**Milestone: Cannot test unless implementation has been completed**	0 days	Thu 15/03/12	Thu 15/03/12	
11	**Testing**	**10 days**	**Thu 15/03/12**	**Wed 28/03/12**	**6**
12	Test database and detail results	5 days	Thu 15/03/12	Wed 21/03/12	
13	Test program and detail results	5 days	Thu 22/03/12	Wed 28/03/12	
14	Advanced	16 days	Sat 24/03/12	Fri 13/04/12	
15	Report Writing	61 days	Mon 23/01/12	Mon 16/04/12	

'12
23 Jan '12
13 Feb '12
05 Mar '12
26 Mar '12
16 Apr
S M T W T F S S M T W T F
15/03



Project: Project Plan
Date: Sun 29/01/12

Task	▬	External Milestone	◆	Manual Summary Rollup	▬
Split	┈	Inactive Task	▭	Manual Summary	▬
Milestone	◆	Inactive Milestone	◇	Start-only	[
Summary	▬	Inactive Summary	▭	Finish-only	]
Project Summary	▬	Manual Task	▬	Deadline	⬇
External Tasks	▭	Duration-only	▬	Progress	▬

Appendix B

Use Case: Sign in

Actors: User

Goal: To log successfully into the system.

Precondition: They know their password.

Summary: User should be able to login successfully by entering the correct username and password.

Related use cases: None

Steps - main flow:

Actor action:	**System Response:**
1. Access login page	
	[Enter correct username and password] 2. Prompts user for username and password
3. Enters details	
	4. User is validated and logged in successfully.

Steps - exceptional flow 1:

Actor action:	**System Response:**
	4. [Username not recognised] Display message and go back to step 2.

Steps - exceptional flow 2:

Actor action:	**System Response:**
	4. [Password not recognised] Display message and go back to step 2.

Use Case: View all students

Actors: User

Goal: View the list of all students

Precondition: User is already logged in and has access to students

Summary: User will view a list of all the students at school

Steps - **main flow**:

Actor action:	**System Response:**
1. Click on view student	
	2. Return a list of all the students

Use Case: View Staff list

Actors: User

Goal: View the list of staff members located at the school

Precondition: User is already logged in and has subscribed to stocks previously

Summary: User will view a list of all the stocks that they have subscribed to

Steps - **main flow**:

Actor action:	**System Response:**
1. Click on view staff	
	3. Return a list of all the staff members.

Use Case: Add student to students database

Actors: User

Goal: Add student to the database.

Precondition: User is already logged in

Summary: User will type in all details and add a student to the database

Related use cases: Find Student

Steps - **main flow**:

Actor action:	**System Response:**
1. Include 'Find Student'	
2. Types details in of student	

	3. [Check If Student Exists] Returns student details for user to confirm
4. User confirms student details	
	1. System updates student database

Steps - **exceptional flow**:

Actor action:	**System Response:**
	5.[If Student exists]. Go to step 2.

Use Case: Remove Student from database

Actors: User

Goal: Remove a stock student from the database

Precondition: User is already logged in

Summary: User will select and remove a student from their student list

Related use cases: Find student

Steps - **main flow**:

Actor action:	**System Response:**
6. Include ‘Find Student	
7. Click on the student and clicks remove button	
	[Check if student exists] 8. Return confirmation check
9. Confirms the removed student	
	10. Confirms the removal and updates the student list.

Steps - **exceptional flow**:

Actor action:	**System Response:**
	4. [If student does not exist]. Return to step 2

Use Case: Find Student
Actors: User
Goal: Finds a student from the database
Precondition: User is already logged in
Summary: User will search for a particular student.
Related use cases: Find Student
Steps - **main flow**:

Actor action:	**System Response:**
1. Include ‘ Find Student’	
2. User searches for a student	
	[Student exists] 3. Returns student details searched by user

Steps - **exceptional flow**:

Actor action:	**System Response:**
	3.[Student does not exist]. Go to Step 2.

Use Case: Update Student/ staff Information
Actors: DBMS, user
Goal: Updates student and informs relevant users
Precondition: Student is already exists in system
Summary: User can update details of student or staff member
Related use cases: Find Staff, Find Student
Steps - **main flow**:

Actor action:	**System Response:**
1. Include ‘ Find Student’ & ‘Find Staff’	
2. User selects either from student or staff whose details they want to update	
	[Check staff/ student exists. 3. Checks details and sends confirmation request.
4. User confirms update	
	5. Student/ staff details

	update in database

Steps - **exceptional flow**:

Actor action:	**System Response:**
	[Student or staff member does not exist] 6. Return to step 2

Appendix C

MYSQL Code for creation of my tables.

```
-- -----------------------------------------------------
-- Table Event
-- -----------------------------------------------------
CREATE  TABLE IF NOT EXISTS Event (
  eventId VARCHAR(10) NOT NULL ,
  subjectId VARCHAR(6) NOT NULL ,
  dateOf DATE NOT NULL ,
  startTime TIME NOT NULL ,
  endTime TIME NOT NULL ,
  PRIMARY KEY (Eventid) ,
  INDEX fkEvent_Subject (subjectId ASC) ,
  CONSTRAINT fkEvent_Subject
    FOREIGN KEY (subjectId )
    REFERENCES Subject (subjectId ))
```

The code above describes how I made my Event table; it represents my Event entity and holds the attributes of eventId, subjectId, dateOf, startTime, endTime. subjectId is a foreign key from another table(Subject).

```
-- -----------------------------------------------------
-- Table Staff
-- -----------------------------------------------------
CREATE  TABLE IF NOT EXISTS Staff (
  staffId VARCHAR(6) NOT NULL DEFAULT     ,
  fName VARCHAR(45) NOT NULL ,
  lName VARCHAR(45) NOT NULL ,
  Title VARCHAR(10) NULL DEFAULT NULL ,
  DeptId VARCHAR(3) NULL DEFAULT NULL ,
  PRIMARY KEY (staffid) ,
  INDEX fk_StaffDept (DeptId ASC) ,
  CONSTRAINT fk_StaffDept
    FOREIGN KEY (DeptId )
    REFERENCES department (deptId))
```

The code above describes how I made my Staff table; it represents my Staff entity and holds the attributes of eventId, subjectId, dateOf, startTime, endTime. subjectId is a foreign key from another table(Subject).

```
-- -----------------------------------------------------
-- Table Student
-- -----------------------------------------------------
CREATE  TABLE IF NOT EXISTS Student (
  studentId VARCHAR(6) NOT NULL DEFAULT,
  fName VARCHAR(45) NOT NULL ,
  lName VARCHAR(45) NOT NULL ,
  dob DATE NULL DEFAULT NULL ,
  address VARCHAR(255) NOT NULL ,
  PRIMARY KEY (`studentid`))
```

The code above describes how I made my Student table; it represents my Student entity and holds the attributes of studentId, first name, last name, age, dob.

-- ---
-- Table Room
-- ---
```
CREATE  TABLE IF NOT EXISTS Room (
  roomId VARCHAR(6) NOT NULL DEFAULT     ,
  rName VARCHAR(45) NOT NULL ,
  capacity INT(11) NOT NULL ,
  PRIMARY KEY (roomid) )
```

The code above describes how I made my Room table; it represents my Room entity and holds the attributes of roomId, rName, capacity.

-- ---
-- Table Equipment
-- ---
```
CREATE  TABLE IF NOT EXISTS Equipment (
  rEquipid VARCHAR(6) NOT NULL ,
  equipname VARCHAR(45) NOT NULL ,
  PRIMARY KEY (rEquipid) )
```
The code above describes how I made my Equipment table; it represents my Equipment entity and holds the attributes of rEquipId, equipName.

-- ---
-- Table Attends
-- ---
```
CREATE  TABLE IF NOT EXISTS Attends (
  attendsId VARCHAR(6) NOT NULL DEFAULT     ,
  studentId VARCHAR(6) NOT NULL ,
  eventId VARCHAR(10) NOT NULL ,
  PRIMARY KEY (attendsid) ,
  INDEX FK_AttendsStudent (studentid ASC) ,
  INDEX FK_AttendsEvent (eventid ASC) ,
  CONSTRAINT FK_AttendsEvent
    FOREIGN KEY (eventid )
    REFERENCES event (Eventid )     ,
  CONSTRAINT FK_AttendsStudent
    FOREIGN KEY (studentid )
    REFERENCES student (studentid ) )
```

This table represents my link entity which makes the relationship between my Event entity and my Student Entity attainable. The entity includes an attendId, studentId and eventID. studentId and eventId are foreign keys from Event and Student.

-- ---
-- Table EquipinRoom
-- ---
```
CREATE  TABLE IF NOT EXISTS EquipinRoom (
  equipid VARCHAR(6) NOT NULL ,
  RoomId VARCHAR(6) NULL DEFAULT NULL ,
  rEquipId VARCHAR(6) NULL DEFAULT NULL ,
  PRIMARY KEY (equipid) ,
  INDEX fkRoom_EquipinRoom (RoomId ASC) ,
  INDEX fkrEquipid_EquipinRoom (rEquipId ASC) ,
  CONSTRAINT fkrEquipid_EquipinRoom
  FOREIGN KEY (rEquipId )
    REFERENCES equipment (rEquipid ),
  CONSTRAINT fkRoom_EquipinRoom
```

```
    FOREIGN KEY (roomId )
    REFERENCES Room (roomid ))
```

This table represents my link entity which makes the relationship between my Equipment entity and my Room Entity manageable. The entity includes an equipId, roomId and rEquipId. roomId and rEquipId are foreign keys from Room and equipment.

```
-- -----------------------------------------------------
-- Table Expertise
-- -----------------------------------------------------
CREATE  TABLE IF NOT EXISTS Expertise (
  expertiseId VARCHAR(6) NOT NULL DEFAULT    ,
  staffId VARCHAR(6) NULL DEFAULT NULL ,
  subjectId VARCHAR(6) NULL DEFAULT NULL ,
  PRIMARY KEY (expertiseId) ,
  INDEX fk_staffExpertise (staffid ASC) ,
  INDEX fk_subjectExpertise (subjectid ASC) ,
  CONSTRAINT fk_staffExpertise
    FOREIGN KEY (staffid )
    REFERENCES Staff (staffid ),
  CONSTRAINT fk_subjectExpertise
    FOREIGN KEY (subjectId )
    REFERENCES Subject (subjectId ))
```

This table represents my link entity which makes the relationship between my Staff entity and my Subject Entity manageable. The entity includes an expertiseId, staffId and subjectId. staffId and subjectId are foreign keys from Staff and Subject.

```
-- -----------------------------------------------------
-- Table Hostedin
-- -----------------------------------------------------
CREATE  TABLE IF NOT EXISTS Hostedin (
  hostId VARCHAR(6) NOT NULL ,
  roomId VARCHAR(6) NOT NULL ,
  eventId VARCHAR(10) NOT NULL ,
  PRIMARY KEY (hostId) ,
  INDEX fk_RoomHostedIn (roomid ASC) ,
  INDEX fk_EventHostedin (Eventid ASC) ,
  CONSTRAINT fk_eventHostedin
    FOREIGN KEY (eventId)
    REFERENCES Event (eventId)
    ON DELETE RESTRICT,
  CONSTRAINT fk_RoomHostedIn
    FOREIGN KEY (roomId)
    REFERENCES Room (roomId )
    ON DELETE RESTRICT)
```

This table represents my link entity which makes the relationship between my Event entity and my Room Entity attainable. The entity includes a hostId, roomId and eventID. roomId and eventId are foreign keys from Event and Room.

```
-- -----------------------------------------------------
-- Table Teaches
-- -----------------------------------------------------
```

```
CREATE TABLE IF NOT EXISTS Teaches (
  teachesId VARCHAR(6) NOT NULL ,
  staffId VARCHAR(6) NOT NULL ,
  eventId VARCHAR(10) NOT NULL ,
  PRIMARY KEY (teachesId),
  INDEX fk_StaffTeaches (staffid ASC),
  INDEX fk_teachesEvent (Eventid ASC),
  CONSTRAINT fk_StaffTeaches
    FOREIGN KEY (staffId)
    REFERENCES Staff (staffId),
  CONSTRAINT fk_teachesEvent
    FOREIGN KEY (eventId)
    REFERENCES Event (eventId))
```

This table represents my link entity which makes the relationship between my Staff entity and my Event Entity manageable. The entity includes a teachesId, staffId and eventID. staffId and eventId are foreign keys from Event and Staff.

Appendix D

```
INSERT INTO Student VALUES
('S1','Amy','Charge','1999-07-08','124 Springfield, Oakshire'),
('S2','Tom','Field','1997-02-04', '11 Cloud Street, Cheshire'),
('S3','Kyle','Banks','1999-12-24','12 Sunshine Avenue, Oakshire'),
('S4','Barry','Stiles','1998-03-02','121 Rain Drive, Cheshire'),
('S5','Joanne','Perry','1996-01-01', '38 Smoke Avenue, Hertfordshire'),
('S6','John','Smith','1996-04-12','11 Robin Hood Meadow, Bedfordshire'),
('S7','Ben','Jones',' 1999-03-17','66 Fire Way, Bedfordshire'),
('S8','Rachel', 'Smith','2000-06-02','99 Thames Meade, Cheshire'),
('S9','James', 'Good',' 1998-03-14','104 Driers Lane, Oakshire'),
('S10','Paul', 'Turner','1997-02-27','220 Oak Drive, Hertfordshire');

INSERT INTO Subject VALUES
('S_01','Maths','D1'),
('S_02','English','D2'),
('S_03','History','D3'),
('S_04','French','D4'),
('S_05','Spanish','D4'),
('S_06','Science','D5'),
('S_07','Drama','D7'),
('S_08','Music','D7'),
('S_09','PE','D6'),
('S_10','Technology','D8');

INSERT INTO Room VALUES
('r1','Room 1','30'),
('r2','Room 2','30'),
('r3','Room 3','30'),
('r4','Room 4' ,'28'),
('r5','Room 5' ,'27'),
('r6','Room 6','25'),
('r7','Room 7','30'),
('r8','Room 8','30'),
('r9','Room 9','30'),
('r10','Room 10','30');

INSERT INTO Equipment VALUES
('1','OHP'),
('2','Projector'),
('3','Extra Whiteboard'),
('4','Laptop');

INSERT INTO Event VALUES
(,E1,S_01,05/03/2012,08:40:00,09:40:00),
(,E2,S_02,05/03/2012,09:40:00,10:40:00),
(,E3,S_03,05/03/2012,11:00:00,12:00:00),
(,E4,S_04,05/03/2012,12:00:00,13:00:00),
(,E5,S_05,05/03/2012,13:40:00,14:40:00),
(,E6,S_06,05/03/2012,14:40:00,15:30:00),
(,E7,S_07,06/03/2012,08:40:00,09:40:00),
(,E8,S_08,06/03/2012,09:40:00,10:40:00),
(,E9,S_09,06/03/2012,11:00:00,12:00:00),
(,E10,S_10,06/03/2012,12:00:00,13:00:00),
(,E11,S_01,06/03/2012,13:40:00,14:40:00),
(,E12,S_02,06/03/2012,14:40:00,15:30:00),
(,E13,S_03,07/03/2012,08:40:00,09:40:00),
(,E14,S_04,07/03/2012,09:40:00,10:40:00),
(,E15,S_05,07/03/2012,11:00:00,12:00:00),
(,E16,S_06,07/03/2012,12:00:00,13:00:00),
(,E17,S_07,07/03/2012,13:40:00,14:40:00),
(,E18,S_08,07/03/2012,14:40:00,15:30:00),
(,E19,S_09,08/03/2012,08:40:00,09:40:00),
(,E20,S_10,08/03/2012,09:40:00,10:40:00),
(,E21,S_01,08/03/2012,11:00:00,12:00:00),
(,E22,S_02,08/03/2012,12:00:00,13:00:00),
(,E23,S_03,08/03/2012,13:40:00,14:40:00),
(,E24,S_04,08/03/2012,14:40:00,15:30:00),
(,E25,S_05,09/03/2012,08:40:00,09:40:00),
(,E26,S_06,09/03/2012,09:40:00,10:40:00),
(,E27,S_07,09/03/2012,11:00:00,12:00:00),
(,E28,S_08,09/03/2012,12:00:00,13:00:00),
(,E29,S_09,09/03/2012,13:40:00,14:40:00),
(,E30,S_10,09/03/2012,14:40:00,15:30:00);
```

Appendix E

Alter Table syntax

```
ALTER TABLE `project`.`equipinroom`  ADD CONSTRAINT `fk_EquipinRoom`
 FOREIGN KEY (`rEquipId` ) REFERENCES `project`.`roomequipment` (`rEquipId` ) ON DELETE
RESTRICT, ADD CONSTRAINT `fk_RoomEquip` FOREIGN KEY (`RoomId` ) REFERENCES
`project`.`room` (`roomid` ) ON DELETE RESTRICT;


ALTER TABLE `project`.`attends`  ADD CONSTRAINT `FK_AttendsStudent` FOREIGN KEY
(`studentId` ) REFERENCES `project`.`student` (`studentid` ) ON DELETE RESTRICT;
 ADD CONSTRAINT `FK_AttendsEvent` FOREIGN KEY (`eventId` ) REFERENCES
`project`.`event` (`eventid` ) ON DELETE RESTRICT;
ALTER TABLE `project`.`event` ADD CONSTRAINT `FK_EventSubject`FOREIGN KEY
(`SubjectId` ) REFERENCES `project`.`subject` (`subjectid` ) ON DELETE RESTRICT;

ALTER TABLE `project`.`expertise`  ADD CONSTRAINT `fk_staffExpertise` FOREIGN KEY
(`staffid` ) REFERENCES `project`.`staff` (`staffid` ) ON DELETE RESTRICT,  ADD CONSTRAINT
`fk_subjectExpertise` FOREIGN KEY (`subjectid` ) REFERENCES `project`.`subject` (`subjectid`
) ON DELETE RESTRICT;
ALTER TABLE `project`.`hostedin`  ADD CONSTRAINT `fk_EventHostedin` FOREIGN KEY
(`eventid` ) REFERENCES `project`.`event` (`eventid` ) ON DELETE RESTRICT,  ADD
CONSTRAINT `fk_RoomHostedIn` FOREIGN KEY (`RoomId` ) REFERENCES `project`.`room`
(`roomid` ) ON DELETE RESTRICT;

ALTER TABLE `project`.`staff`  ADD CONSTRAINT `fk_StaffDept` FOREIGN KEY (`DeptId` )
REFERENCES `project`.`department` (`deptid` ) ON DELETE RESTRICT;


ALTER TABLE `project`.`subject`  ADD CONSTRAINT `fk_SubjectDept` FOREIGN KEY (`DeptId`
) REFERENCES `project`.`department` (`deptid` ) ON DELETE RESTRICT;



ALTER TABLE `project`.`teaches`  ADD CONSTRAINT `fk_StaffTeaches` FOREIGN KEY (`staffId`
) REFERENCES `project`.`staff` (`staffid` ) ON DELETE RESTRICT,  ADD CONSTRAINT
`fk_teachesEvent` FOREIGN KEY (`eventId` ) REFERENCES `project`.`event` (`eventid` ) ON
DELETE RESTRICT;
```

Appendix F

Queries:

This query returns a list of which equipment is located in which room.

```
SELECT * FROM (
    SELECT roomid, rEquipId FROM equipinroom
    ) d
    INNER JOIN (
    SELECT EquipName, rEquipId FROM equipment
    ) e
    ON d.rEquipId = e.rEquipId
```

Returns eventid with list of student names with events they're registered, ordered by their last name:

```
select student.studentid, student.fname, student.lname,
attends.eventid
from student
left join attends
on student.studentid = attends.studentid
order by student.lname
```

Returns which equipment is at all events

```
select hostedin.eventid, equipment.equipname
from equipment
left join hostedin
on equipment.requipid = equipment.requipid
```

Query to select and individual student and which events they are registered to.

```
select student.studentid, student.fname, attends.eventid
from student
inner join attends
on student.studentid = attends.studentid
where student.studentid = 's1'
```

Query to select all students and show which events they are registered to.

```
SELECT * from student
inner join (
attends INNER JOIN event
ON attends.eventID = event.eventID
)
ON student.studentID = attends.studentID
```

Query to select which rooms equipment is located in.

```
SELECT * FROM (
    SELECT roomid, rEquipId FROM equipinroom
    ) d
    INNER JOIN (
    SELECT EquipName, rEquipId FROM equipment
    ) e
    ON d.rEquipId = e.rEquipId
```

Appendix G

Test Data:

English
History
French
Spanish
Science
Drama
Music
PE
Technology

Ryan	Reynolds	Mr
Sally	Payne	Miss
Phillip	Cole	Mr
Simon	Grant	Mr
Alice	White	Mrs
Trevor	Baker	Mr
Helen	Norris	Mrs
Zoe	Waters	Miss
Amy	Adams	Miss
Todd	Gray	Mr

Tom Field 1997-02-04
Kyle Banks 1999-12-24
Barry Stiles 1998-03-02
Joanne Perry 1996-01-01
John Smith 1996-04-12
Ben Jones 1999-03-17
Rachel Smith 2000-06-02
James Good 1998-03-14
Paul Turner 1997-02-27
Amy Charge 1999-07-08

Appendix H

Program Code

VIEW CLASS

```
/*


 *This class is the design of my user interface, it includes the
creation of my tables, and buttons
 * and it also includes the methods of how i would edit/add to my
database.  The methods in this class were created by me.
 */
/*
 * View.java
 *
 * Created on 22-Mar-2012, 20:53:31
 */
// the following list includes all my imports which i needed for
certain methods to work
package GUI;
import Ctrl.exceptions.NonexistentEntityException;
import Ctrl.exceptions.PreexistingEntityException;
import Entity.myEntityManager;
import DBManage.Student;
import java.util.Iterator;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JOptionPane;
import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;
import javax.swing.table.DefaultTableModel;


/**
 *
 * @author Vicky
 */
// I chose to use a JDialog as i felt it was the best option for me
public class View extends javax.swing.JDialog {

    /** The following code creates a new form called View */
    public View(java.awt.Frame parent, boolean modal) {
        super(parent, modal);
        initComponents();
        fillDate();
        // tracking table selection



studtbl.getSelectionModel().addListSelectionListener(
            new ListSelectionListener() {
            @Override
            // this method was added to attempt to edit the table
held in my database at the final stage of my
            //project it wasn't working
```

```
                public void valueChanged(ListSelectionEvent e) {
                    firePropertyChange("recordSelected",
!isRecordSelected(), isRecordSelected());
                }
            });

    }

    //this is another method that isn't working get, i wanted to be
able to select a student in the table
    // and have their details appear in the textfields below
public boolean isRecordSelected() {
        return studtbl.getSelectedRow() != -1;
    }


          /** This method is called from within the constructor to
//     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method
is
     * always regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        DBManagePUEntityManager = java.beans.Beans.isDesignTime() ?
null :
javax.persistence.Persistence.createEntityManagerFactory("DBManagePU
").createEntityManager();
        studentQuery = java.beans.Beans.isDesignTime() ? null :
DBManagePUEntityManager.createQuery("SELECT s FROM Student s");
        studentList = java.beans.Beans.isDesignTime() ?
java.util.Collections.emptyList() : studentQuery.getResultList();
        jPanel1 = new javax.swing.JPanel();
        jScrollPane1 = new javax.swing.JScrollPane();
        studtbl = new javax.swing.JTable();
        jLabel1 = new javax.swing.JLabel();
        jLabel2 = new javax.swing.JLabel();
        jLabel3 = new javax.swing.JLabel();
        jLabel4 = new javax.swing.JLabel();
        jLabel5 = new javax.swing.JLabel();
        txtid = new javax.swing.JTextField();
        txtfname = new javax.swing.JTextField();
        txtlname = new javax.swing.JTextField();
        txtdob = new javax.swing.JTextField();
        txtaddress = new javax.swing.JTextField();
        bttnAdd = new javax.swing.JButton();
        clearbtn = new javax.swing.JButton();
        deleteButton = new javax.swing.JButton();
        editbtn = new javax.swing.JButton();
        jPanel3 = new javax.swing.JPanel();
        jScrollPane2 = new javax.swing.JScrollPane();
        searchbtn = new javax.swing.JButton();
        txtstuID = new javax.swing.JTextField();
        jTextField2 = new javax.swing.JTextField();
        jMenuBar1 = new javax.swing.JMenuBar();
        exitmenu = new javax.swing.JMenu();
        jMenuItem3 = new javax.swing.JMenuItem();
        jMenu2 = new javax.swing.JMenu();
        jMenu3 = new javax.swing.JMenu();
        jMenuItem2 = new javax.swing.JMenuItem();
        jMenuItem1 = new javax.swing.JMenuItem();


setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOS
E);
        getContentPane().setLayout(new java.awt.CardLayout());

        studtbl.setModel(new javax.swing.table.DefaultTableModel(
            new Object [][] {

            },
            new String [] {
                "Student ID", "First Name", "Last Name", "Date of
Birth", "Address"
            }
        ));
        jScrollPane1.setViewportView(studtbl);

        jLabel1.setText("Student Id");

        jLabel2.setText("First Name");

        jLabel3.setText("Last Name");

        jLabel4.setText("DOB");

        jLabel5.setText("Address");

        bttnAdd.setText("Add");
        bttnAdd.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent
evt) {
                bttnAddActionPerformed(evt);
            }
        });

        clearbtn.setText("Clear");
        clearbtn.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent
evt) {
                clearbtnActionPerformed(evt);
            }
        });

        deleteButton.setText("Delete");

        editbtn.setText("Edit");
        editbtn.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent
evt) {
                editbtnActionPerformed(evt);
            }
        });

        javax.swing.GroupLayout jPanel1Layout = new
javax.swing.GroupLayout(jPanel1);
```

```
        jPanel1.setLayout(jPanel1Layout);
        jPanel1Layout.setHorizontalGroup(

jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.
LEADING)
            .addGroup(jPanel1Layout.createSequentialGroup()

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.
Alignment.LEADING)
                    .addGroup(jPanel1Layout.createSequentialGroup()

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.
Alignment.LEADING)

.addGroup(jPanel1Layout.createSequentialGroup()
                                .addGap(13, 13, 13)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.
Alignment.TRAILING)

.addGroup(jPanel1Layout.createSequentialGroup()

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.
Alignment.TRAILING)
                                            .addComponent(jLabel2)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.
Alignment.LEADING)

.addComponent(jLabel4)

.addComponent(jLabel3)

.addComponent(jLabel5)))
                                        .addGap(18, 18, 18))
                                    .addComponent(bttnAdd)))

.addGroup(jPanel1Layout.createSequentialGroup()
                                .addContainerGap()
                                .addComponent(jLabel1)))

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.
Alignment.LEADING)

.addGroup(jPanel1Layout.createSequentialGroup()
                                .addGap(9, 9, 9)
                                .addComponent(editbtn,
javax.swing.GroupLayout.PREFERRED_SIZE, 60,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                .addGap(10, 10, 10)
                                .addComponent(clearbtn)
                                .addGap(18, 18, 18)
                                .addComponent(deleteButton))
                            .addComponent(txtaddress,
javax.swing.GroupLayout.PREFERRED_SIZE, 51,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.
Alignment.TRAILING, false)
                                .addComponent(txtid,
javax.swing.GroupLayout.Alignment.LEADING)
```

```
                                .addComponent(txtfname,
javax.swing.GroupLayout.Alignment.LEADING)
                                .addComponent(txtlname,
javax.swing.GroupLayout.Alignment.LEADING,
javax.swing.GroupLayout.DEFAULT_SIZE, 50, Short.MAX_VALUE)
                                .addComponent(txtdob,
javax.swing.GroupLayout.Alignment.LEADING))))
                    .addGroup(jPanel1Layout.createSequentialGroup()
                        .addContainerGap()
                        .addComponent(jScrollPane1,
javax.swing.GroupLayout.DEFAULT_SIZE, 438, Short.MAX_VALUE)))
                .addContainerGap())
        );
        jPanel1Layout.setVerticalGroup(

jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.
LEADING)
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jScrollPane1,
javax.swing.GroupLayout.PREFERRED_SIZE, 139,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATE
D)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.
Alignment.BASELINE)
                    .addComponent(txtid,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jLabel1))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATE
D)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.
Alignment.BASELINE)
                    .addComponent(jLabel2)
                    .addComponent(txtfname,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addGap(11, 11, 11)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.
Alignment.BASELINE)
                    .addComponent(jLabel3)
                    .addComponent(txtlname,
javax.swing.GroupLayout.PREFERRED_SIZE, 20,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.
Alignment.BASELINE)
                    .addComponent(txtdob,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
```

```
                    .addComponent(jLabel4))
                .addGap(8, 8, 8)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.
Alignment.BASELINE)
                    .addComponent(txtaddress,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jLabel5))
                .addGap(18, 18, 18)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.
Alignment.BASELINE)
                    .addComponent(bttnAdd)
                    .addComponent(editbtn,
javax.swing.GroupLayout.PREFERRED_SIZE, 23,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(clearbtn)
                    .addComponent(deleteButton))
                .addContainerGap(95, Short.MAX_VALUE))
        );

        getContentPane().add(jPanel1, "card2");

        searchbtn.setText("Search");

        jTextField2.setText("jTextField2");

        javax.swing.GroupLayout jPanel3Layout = new
javax.swing.GroupLayout(jPanel3);
        jPanel3.setLayout(jPanel3Layout);
        jPanel3Layout.setHorizontalGroup(

jPanel3Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.
LEADING)
            .addGroup(jPanel3Layout.createSequentialGroup()

.addGroup(jPanel3Layout.createParallelGroup(javax.swing.GroupLayout.
Alignment.LEADING)
                    .addGroup(jPanel3Layout.createSequentialGroup()
                        .addGap(26, 26, 26)
                        .addComponent(jScrollPane2,
javax.swing.GroupLayout.PREFERRED_SIZE, 255,
javax.swing.GroupLayout.PREFERRED_SIZE))
                    .addGroup(jPanel3Layout.createSequentialGroup()
                        .addGap(39, 39, 39)

.addGroup(jPanel3Layout.createParallelGroup(javax.swing.GroupLayout.
Alignment.TRAILING, false)
                            .addComponent(txtstuID)
                            .addComponent(jTextField2))
                        .addGap(158, 158, 158)
                        .addComponent(searchbtn)))
                .addContainerGap(137, Short.MAX_VALUE))
        );
        jPanel3Layout.setVerticalGroup(

jPanel3Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.
LEADING)
            .addGroup(jPanel3Layout.createSequentialGroup()
                .addGap(29, 29, 29)
                .addComponent(jScrollPane2,
javax.swing.GroupLayout.PREFERRED_SIZE, 125,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addGap(46, 46, 46)

.addGroup(jPanel3Layout.createParallelGroup(javax.swing.GroupLayout.
Alignment.BASELINE)
                    .addComponent(searchbtn)
                    .addComponent(txtstuID,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATE
D)
                .addComponent(jTextField2,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap(179, Short.MAX_VALUE))
        );

        getContentPane().add(jPanel3, "card3");

        exitmenu.setText("File");

        jMenuItem3.setText("Exit");
        jMenuItem3.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent
evt) {
                jMenuItem3ActionPerformed(evt);
            }
        });
        exitmenu.add(jMenuItem3);

        jMenuBar1.add(exitmenu);

        jMenu2.setText("Edit");
        jMenuBar1.add(jMenu2);

        jMenu3.setText("Student");

        jMenuItem2.setText("View");
        jMenuItem2.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent
evt) {
                jMenuItem2ActionPerformed(evt);
            }
        });
        jMenu3.add(jMenuItem2);

        jMenuItem1.setText("Edit");
        jMenu3.add(jMenuItem1);

        jMenuBar1.add(jMenu3);

        setJMenuBar(jMenuBar1);
        pack();
    }// </editor-fold>

    private void bttnAddActionPerformed(java.awt.event.ActionEvent
evt) {
        try {
            // TODO add your handling code here:
     Add();
        } catch (PreexistingEntityException ex) {
            Logger.getLogger(View.class.getName()).log(Level.SEVERE,
null, ex);
        } catch (Exception ex) {
            Logger.getLogger(View.class.getName()).log(Level.SEVERE,
null, ex);
        }

    }

    private void clearbtnActionPerformed(java.awt.event.ActionEvent
evt) {
        // TODO add your handling code here:
        clear();
    }

    private void editbtnActionPerformed(java.awt.event.ActionEvent
evt) {
        try {
            // TODO add your handling code here:
            EditStudent();
        } catch (NonexistentEntityException ex) {
            Logger.getLogger(View.class.getName()).log(Level.SEVERE,
null, ex);
        }
    }

    private void
jMenuItem3ActionPerformed(java.awt.event.ActionEvent evt) {
    System.exit(0);

        // TODO add your handling code here:
    }

    private void
jMenuItem2ActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:




    }

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        /* Set the Nimbus look and feel */
        //<editor-fold defaultstate="collapsed" desc=" Look and feel
setting code (optional) ">
        /* If Nimbus (introduced in Java SE 6) is not available,
stay with the default look and feel.
```

```
         * For details see 
http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.
html 
         */
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info : 
javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {

javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(View.class.getName()).log(java.ut
il.logging.Level.SEVERE, null, ex);
        } catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(View.class.getName()).log(java.ut
il.logging.Level.SEVERE, null, ex);
        } catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(View.class.getName()).log(java.ut
il.logging.Level.SEVERE, null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(View.class.getName()).log(java.ut
il.logging.Level.SEVERE, null, ex);
        }
        //</editor-fold>

        /* Create and display the dialog */
        java.awt.EventQueue.invokeLater(new Runnable() {
// the following method deals with the running of the application, 
and 
            @Override
            public void run() {
                View dialog = new View(new javax.swing.JFrame(), 
true);
                dialog.addWindowListener(new 
java.awt.event.WindowAdapter() {

                    @Override
                    // the closing of the application via the red x 
button
            // at the top right of the screen
                    public void 
windowClosing(java.awt.event.WindowEvent e) {
                        System.exit(0);
                    }
                });
                dialog.setVisible(true);
            }
        });
    }

    // Variables declaration - do not modify
    private javax.persistence.EntityManager DBManagePUEntityManager;
    private javax.swing.JButton bttnAdd;
    private javax.swing.JButton clearbtn;
    private javax.swing.JButton deleteButton;
    private javax.swing.JButton editbtn;
    private javax.swing.JMenu exitmenu;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JMenu jMenu2;
    private javax.swing.JMenu jMenu3;
    private javax.swing.JMenuBar jMenuBar1;
    private javax.swing.JMenuItem jMenuItem1;
    private javax.swing.JMenuItem jMenuItem2;
    private javax.swing.JMenuItem jMenuItem3;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JPanel jPanel3;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JScrollPane jScrollPane2;
    private javax.swing.JTextField jTextField2;
    private javax.swing.JButton searchbtn;
    private java.util.List<DBManage.Student> studentList;
    private javax.persistence.Query studentQuery;
    private javax.swing.JTable studtbl;
    private javax.swing.JTextField txtaddress;
    private javax.swing.JTextField txtdob;
    private javax.swing.JTextField txtfname;
    private javax.swing.JTextField txtid;
    private javax.swing.JTextField txtlname;
    private javax.swing.JTextField txtstuID;
    // End of variables declaration

     /**
     *this method retrieves the students from the database and for
every student
     * it creates a student object and adds it to the rowData which
is then added to
     * the DefaultTabelModel dtm
     *
     */
private void fillDate() {
DefaultTableModel dtm =(DefaultTableModel) studtbl.getModel();
dtm.setRowCount(0);
List<Student> students =
myEntityManager.getStudentJpaController().findStudentEntities();
for(Iterator<Student> it = students.iterator(); it.hasNext();){
Student student = it.next();

Object [] rowData ={student.getStudentid(),student.getFName()
,student.getLName(),student.getDob(),student.getAddress()};
dtm.addRow(rowData);

}
}
// this method clears the text from my text fields
private void clear() {
txtid.setText("");
txtfname.setText("");
txtlname.setText("");
txtdob.setText("");
txtaddress.setText("");
}
```

```
/**
*this method adds what is located in the textfields into my student
database
*using the Create student found in my JPA controller
* it also uses the method FillData()which updates
* the table located on the user interface with the new object thats
been created
*/
//
//
private void Add() throws PreexistingEntityException, Exception {
Student student = new Student();
student.setStudentid(txtid.getText().trim());
student.setFName(txtfname.getText().trim());
student.setLName(txtlname.getText().trim()) ;
student.setAddress(txtaddress.getText().trim());
String studentid = txtid.getText().trim();
String fname = txtfname.getText().trim();
String lname = txtlname.getText().trim();
String address = txtid.getText().trim();
// student.setDob(new Date() = (Date)txtdob.getValue());


if(studentid.equals("")){
JOptionPane.showMessageDialog(this, "Student ID Required");
txtid.requestFocus();
return;
}



if(fname.equals("")){
JOptionPane.showMessageDialog(this, "First Name Required");
txtfname.requestFocus();
return;
}
if(lname.equals("")){
JOptionPane.showMessageDialog(this, "Last Name Required");
txtlname.requestFocus();
return;
}
if(address.equals("")){
JOptionPane.showMessageDialog(this, "Address Required");
txtaddress.requestFocus();
return;
}


System.out.println (student.getStudentid());
myEntityManager.getStudentJpaController().create(student);
clear();
fillDate();

    }
/**
*
*this is my edit student method, at this time it can edit a student
that exists, but if it edits a student that doesn't
* exist it creates a new student (which is incorrect behaviour)
*/
//
```

```
private void EditStudent() throws NonexistentEntityException {
    Student student = new Student();
    student.setStudentid(txtid.getText().trim());
    student.setFName(txtfname.getText().trim());
    student.setLName(txtlname.getText().trim()) ;
    student.setAddress(txtaddress.getText().trim());
    try {
        myEntityManager.getStudentJpaController().edit(student);
    } catch (Exception ex) {
        Logger.getLogger(View.class.getName()).log(Level.SEVERE,
null, ex);
    }
//
//          if (findStudent(id) !=  null) {
//               throw new PreexistingEntityException("Student " +
student + " already exists.", ex);
//           }
//

   clear();
   fillDate();

}}
```

Test Class:

```
package School;

import DBManage.Student;

import Entity.myEntityManager;

import java.util.Iterator;

import java.util.List;



/**
 *
 * @author Vicky
 */
public class School {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) throws Exception {
        // TODO code application logic here
        m1();

    }

    private static void m1() throws Exception  {

   Student student = new Student();
student.setStudentid("s12");
student.setFName("Helen");
student.setLName("Green");
student.setAddress("11 Space Avenue,Cheshire ");
System.out.println();


 myEntityManager.getStudentJpaController().create(student);
    }}
```

Student Class:

```
package DBManage;

import java.io.Serializable;
import java.util.Date;
import javax.persistence.Basic;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.Table;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;
import javax.xml.bind.annotation.XmlRootElement;

/**
 *
 * @author Vicky
 */
@Entity
@Table(name = "student", catalog = "project", schema = "")
@XmlRootElement
@NamedQueries({
    @NamedQuery(name = "Student.findAll", query = "SELECT s FROM
Student s"),
    @NamedQuery(name = "Student.findByStudentid", query = "SELECT s
FROM Student s WHERE s.studentid = :studentid"),
    @NamedQuery(name = "Student.findByFName", query = "SELECT s FROM
Student s WHERE s.fName = :fName"),
    @NamedQuery(name = "Student.findByLName", query = "SELECT s FROM
Student s WHERE s.lName = :lName"),
    @NamedQuery(name = "Student.findByDob", query = "SELECT s FROM
Student s WHERE s.dob = :dob"),
    @NamedQuery(name = "Student.findByAddress", query = "SELECT s
FROM Student s WHERE s.address = :address")})
public class Student implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @Basic(optional = false)
    @Column(name = "studentid", nullable = false, length = 6)
    private String studentid;
    @Basic(optional = false)
    @Column(name = "fName", nullable = false, length = 45)
    private String fName;
    @Basic(optional = false)
    @Column(name = "lName", nullable = false, length = 45)
    private String lName;
    @Column(name = "dob")
    @Temporal(TemporalType.DATE)
    private Date dob;
    @Basic(optional = false)
    @Column(name = "address", nullable = false, length = 255)
    private String address;
    public Student() {
    }

    public Student(String studentid) {
        this.studentid = studentid;
    }

    public Student(String studentid, String fName, String lName,
String address) {
        this.studentid = studentid;
        this.fName = fName;
        this.lName = lName;
        this.address = address;
    }

    public String getStudentid() {
        return studentid;
    }

    public void setStudentid(String studentid) {
        this.studentid = studentid;
    }

    public String getFName() {
        return fName;
    }

    public void setFName(String fName) {
        this.fName = fName;
    }

    public String getLName() {
        return lName;
    }

    public void setLName(String lName) {
        this.lName = lName;
    }

    public Date getDob() {
        return dob;
    }

    public void setDob(Date dob) {
        this.dob = dob;
    }

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    @Override
    public int hashCode() {
        int hash = 0;
        hash += (studentid != null ? studentid.hashCode() : 0);
        return hash;
    }

    @Override
    public boolean equals(Object object) {
        // TODO: Warning - this method won't work in the case the id
fields are not set
        if (!(object instanceof Student)) {
            return false;
        }
        Student other = (Student) object;
        if ((this.studentid == null && other.studentid != null) ||
(this.studentid != null && !this.studentid.equals(other.studentid)))
{
            return false;
        }
        return true;
    }

    @Override
    public String toString() {
        return "DBManage.Student[ studentid=" + studentid + " ]";
    }

}
```

Student JPA Controller

```
import Ctrl.exceptions.NonexistentEntityException;
import Ctrl.exceptions.PreexistingEntityException;
import DBManage.Student;
import java.io.Serializable;
import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Query;
import javax.persistence.EntityNotFoundException;
import javax.persistence.criteria.CriteriaQuery;
import javax.persistence.criteria.Root;

/**
 *
 * @author Vicky
 * this class is auto generated by Netbeans, it allows me to persist
my database
 */
public class StudentJpaController implements Serializable {

    public StudentJpaController(EntityManagerFactory emf) {
        this.emf = emf;
    }
    private EntityManagerFactory emf = null;

    public EntityManager getEntityManager() {
        return emf.createEntityManager();
    }

    public void create(Student student) throws
PreexistingEntityException, Exception {
        EntityManager em = null;
        try {
            em = getEntityManager();
            EntityTransaction t = em.getTransaction();
            t.begin();
            em.persist(student);
            t.commit();
        } catch (Exception ex) {
              String id = student.getStudentid();
            if (findStudent(id) !=  null) {
                throw new PreexistingEntityException("Student " +
student + " already exists.", ex);
            }
            throw ex;
        } finally {
            if (em != null) {
                em.close();
            }
        }
    }

    public void edit(Student student) throws
NonexistentEntityException, Exception {

        EntityManager em = null;
        try {
            em = getEntityManager();
            em.getTransaction().begin();
            student = em.merge(student);
            em.getTransaction().commit();
        } catch (Exception ex) {
            String msg = ex.getLocalizedMessage();
              String id = student.getStudentid();
            if (msg == null || msg.length() == 0) {
                   if (findStudent(id) == null) {
                    throw new NonexistentEntityException("The
student with id " + id + " no longer exists.");
                }

            throw ex;
            }} finally {
            if (em != null) {
                em.close();
            }
        }
    }

    public void destroy(String id) throws NonexistentEntityException
{
        EntityManager em = null;
        try {
            em = getEntityManager();
            em.getTransaction().begin();
            Student student;
            try {
                student = em.getReference(Student.class, id);
                student.getStudentid();
            } catch (EntityNotFoundException enfe) {
                throw new NonexistentEntityException("The student
with id " + id + " no longer exists.", enfe);
            }
            em.remove(student);
            em.getTransaction().commit();
        } finally {
            if (em != null) {
                em.close();
            }
        }
    }

    public List<Student> findStudentEntities() {
        return findStudentEntities(true, -1, -1);
    }

    public List<Student> findStudentEntities(int maxResults, int
firstResult) {
        return findStudentEntities(false, maxResults, firstResult);
    }

    private List<Student> findStudentEntities(boolean all, int
maxResults, int firstResult) {
        EntityManager em = getEntityManager();
        try {
            CriteriaQuery cq =
em.getCriteriaBuilder().createQuery();
            cq.select(cq.from(Student.class));
            Query q = em.createQuery(cq);
            if (!all) {
                q.setMaxResults(maxResults);
                q.setFirstResult(firstResult);
            }
            return q.getResultList();
        } finally {
            em.close();
        }
    }

    public Student findStudent(String studentid) {
        EntityManager em = getEntityManager();

        try {
            return em.find(Student.class, studentid);

        } finally {
            em.close();
        }
    }

    public int getStudentCount() {
        EntityManager em = getEntityManager();
        try {
            CriteriaQuery cq =
em.getCriteriaBuilder().createQuery();
            Root<Student> rt = cq.from(Student.class);
            cq.select(em.getCriteriaBuilder().count(rt));
            Query q = em.createQuery(cq);
            return ((Long) q.getSingleResult()).intValue();
        } finally {
            em.close();}}}
```