**4.1. Introduction**

The goal of this chapter is to outline and examine the design considerations and processes carried out based on the results of analysis. This chapter will examine both the database design process including relations and schemas as well as the user interface design process including its layout and structure.

Design can be compared to an instruction book for flat pack furniture. It provides diagrams and guidance of how the elements will fit together, however, much the same as flat pack furniture, things can be missing and elements will need to be added or modified to achieve the final desired result. It is therefore understood that the design processes used will ensure a good outcome but changes may still be needed further in development because of unforeseen circumstances. This idea falls in line with the prototype methodology defined in section 3.2.

**4.2. Database Design**

Database design is the process of producing a detailed data model of a database. This logical data model contains all the needed logical and physical design choices and physical storage parameters needed to generate a design in a Data Definition Language, which can then be used to create a database. A fully attributed data model contains detailed attributes for each entity.

The term database design can be used to describe many different parts of the design of an overall database system. Principally, and most correctly, it can be thought of as the logical design of the base data structures used to store the data. In the relational model these are the tables and views. In an object database the entities and relationships map directly to object classes and named relationships. However, the term database design could also be used to apply to the overall process of designing, not just the base data structures, but also the forms and queries used as part of the overall database application within the database management system (DBMS).

The process of doing database design generally consists of a number of steps which were fully observed while designing the database for this University Information Management System. Namely:

* Determined the relationships between the different data elements.
* Superimposed a logical structure upon the data on the basis of these relationships.

Both of these stages were utilized in the design of the database system.

**4.3. Types of Database Design**

**4.3.1 Conceptual schema**

Once a database designer is aware of the data which is to be stored within the database, they must then determine where dependency is within the data. Sometimes when data is changed you can be changing other data that is not visible. For example, in a list of names and addresses, assuming a situation where multiple people can have the same address, but one person cannot have more than one address; the name is dependent upon the address, because if the address is different, then the associated name is different too. However, the other way around is different. One attribute can change and not another.

(NOTE: A common misconception is that the relational model is so called because of the stating of relationships between data elements therein. This is not true. The relational model is so named because it is based upon the mathematical structures known as relations.)

**4.3.2 Logically structuring data**

Once the relationships and dependencies amongst the various pieces of information have been determined, it is possible to arrange the data into a logical structure which can then be mapped into the storage objects supported by the database management system. In the case of relational databases the storage objects are tables which store data in rows and columns.

Each table may represent an implementation of either a logical object or a relationship joining one or more instances of one or more logical objects. Relationships between tables may then be stored as links connecting child tables with parents. Since complex logical relationships are themselves tables they will probably have links to more than one parent.

In an Object database the storage objects correspond directly to the objects used by the Object-oriented programming language used to write the applications that will manage and access the data. The relationships may be defined as attributes of the object classes involved or as methods that operate on the object classes.

**4.3.2 Physical database design**

The physical design of the database specifies the physical configuration of the database on the storage media. This includes detailed specification of data elements, data types, indexing options and other parameters residing in the DBMS data dictionary. It is the detailed design

Note that all of these steps of database design types were at every point of this development observed to ensure conformity with the database design standard at every stage while designing the University Information Management System.

**4.4 The Design Process**

The design process used consists of the following steps

1. Determine the purpose of your database - This was a major milestone because it helps prepared steps required in the database design.
2. Find and organize the information required - Gathered all of the types of information needed to be recorded in the database, such as student biodata, emails, phone numbers etc.
3. Divide the information into tables – This help divide all information items into major entities or subjects, such as Biodata, Faculties, Departments, Programs, and Courses etc. Each subject then becomes a table.
4. Turned information items into columns - Decides what information to be stored in each table. Each item becomes a field, and is displayed as a column in the table. For example, a Biodata table includes fields such as Surname, Middle Name Date of birth etc.
5. Specify primary keys - Choose each table’s primary key. The primary key is a column that is used to uniquely identify each row. An example is the Code field.
6. Set up the table relationships - Decide how the data in one table is related to the data in other tables. Various fields were added to tables to clarify the relationships, as necessary.
7. Refine design - Analyzed design for errors. Create the tables and add a few records of sample data. See if you can get the results you want from your tables. Make adjustments to the design, as needed.
8. Apply the normalization rules - Apply the data normalization rules to see if your tables are structured correctly. Make adjustments to the tables

**4.5. Database Normalization**

In the design of a relational database management system (RDBMS), the process of organizing data to minimize redundancy is called normalization. The goal of database normalization is to decompose relations with anomalies in order to produce smaller, well-structured relations. Normalization usually involves dividing large tables into smaller (and less redundant) tables and defining relationships between them. The objective is to isolate data so that additions, deletions, and modifications of a field can be made in just one table and then propagated through the rest of the database via the defined relationships.

Edgar F. Codd, the inventor of the relational model, introduced the concept of normalization and what we now know as the First Normal Form (1NF) in 1970. Codd went on to define the Second Normal Form (2NF) and Third Normal Form (3NF) in 1971, and Codd and Raymond F. Boyce defined the Boyce-Codd Normal Form (BCNF) in 1974. Higher normal forms were defined by other theorists in subsequent years, the most recent being the Sixth Normal Form (6NF) introduced by Chris Date, Hugh Darwen, and Nikos Lorentzos in 2002.

Informally, a relational database table (the computerized representation of a relation) is often described as "normalized" if it is in the Third Normal Form. Most 3NF tables are free of insertion, update, and deletion anomalies, i.e. in most cases 3NF tables adhere to BCNF, 4NF, and 5NF (but typically not 6NF).

This data set has been normalized to third normal form and the processes used for achieving this are not outlined in this section due to time constraint allocated to the project (*See Entity Diagram - Appendix 1a – 1g for the third normalization tables*).

**4.5.1 Entity-relationship model**

In software engineering, an entity-relationship model (ERM) is an abstract and conceptual representation of data. Entity-relationship modeling is a database modeling method, used to produce a type of conceptual schema or semantic data model of a system, often a relational database, and its requirements in a top-down fashion. Diagrams created by this process are called entity-relationship diagrams, ER diagrams, or ERDs. (*http://en.wikipedia.org/wiki/Entity-relationship\_model*). (*See Entity Diagram - Appendix 1a – 1g design for the university information management system for details*)

**4.6 Database Schema**

A database schema (pronounced skee-ma, /ˈski.mə/) of a database system is its structure described in a formal language supported by the database management system (DBMS) and refers to the organization of data to create a blueprint of how a database will be constructed (divided into database tables).

Based on the above mention capability of database schema that the divided database tables also referred to as entities with likes activities in this project were grouped into their respective schemas. E.g. Addresses, BioDatas, Emails, Emergencies are all grouped under a schema called **[SetUp]**.

**4.7 Naming Conventions**

There were quiet a few simple rules I adopted naming all the objects used in the database (including the name of the database itself):

1. **Use of spaces in object names was avoided**. It might seem tempting and cute to have a table named "Bio Data" or “Medical Conditions” but this is a nightmare to deal with programmatically. DTS, for example, will have serious problems, and anyone manually referencing the table will have to enclose it in [square brackets] to avoid an error.
2. **Using reserved words was quiet avoided**. This is easier said than done, as there is a very large list of current and future reserved words to check against. Even if the word used doesn't cause an error, it will still cause confusion when someone is editing code and Query Analyzer applies color highlighting to table or column names.
3. **Use of dashes in database names was avoided**. Several SQL Server helper tools will complain about this; even the tools included with SQL Server 2005 are still unable to access such databases in certain cases.
4. Started all object names with a letter not a number
5. Keeping names of database objects meaningful
6. **Used sensible case**. There is little worse than going through a database schema where all the table names are in ALL CAPS—I feel like SQL Server is yelling at me! Likewise, trying to read a procedure name like "spfinalsignatoriesdelete" could drive some people to drink or go gaga. So Pascal-casing naming convention was used instead. E.g. SPFinalSignatoriesDelete.
7. **Hungarian notation was avoided**. The name of the object should make it pretty obvious what type of data it contains, and if for some reason it does not, and then there are always the metadata tables and/or the documentation you should have written when designing the system. Using datatype-style prefixes for columns like ICode (integer) and VEmail (varchar) not only make the column names harder to read, they also make them less flexible. Let's say ICode exceeds the upper bound of the Integer data type, and we need to convert it to a BigInt. How do we do that? Do we change the column name and—simultaneously—all the code in all of our projects to reference the new name? Do we create a view with the old column name, rename the base table and give its name to the view? Do we leave the column name the same, instantly making the convention obsolete? This borrowed concept from programming languages is not really necessary in the confines of a database, where we are dealing with much fewer variables, and data structures are fixed.
8. Use of [Schema].[ObjectName] was used. You will only be bitten by this once or twice until you make it a fundamental part of your database practices. When you are logged in as a non-dbo user, and you create a table without giving it an owner prefix [Schema], e.g.:

CREATE TABLE foo

(

Bar INT

)

What can happen is that other users won't see it, because it is stored in the system as you.foo instead of dbo.foo or them.foo. If you consistently use the [Schema].[ObjectName], you will eliminate the possibility of creating the same object name twice, with different owner names. This can be a mess, especially when moving the database to a new server. Note that not specifying the owner/schema prefix when accessing objects in T-SQL queries will cause more work for the engine, as it will have to figure out WHICH table named "foo" you're talking about.

1. **Tables / Views**: Tables are sets of entities in the data model; as such, their names were given in plural/collective nouns.

CREATE TABLE [SetUp].[Countries](

[Code] [nvarchar](50) NOT NULL,

[Name] [nvarchar](256) NOT NULL,

CONSTRAINT [PK\_Countries] PRIMARY KEY CLUSTERED

(

[Code] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON) ON [PRIMARY]

) ON [PRIMARY]

1. **Columns**: column names represent an attribute of the entity in our data model; so, they are singular, and they reflect the entity, not the table name.

CREATE TABLE [Personals].[BioDatas](

[Code] [nvarchar](50) NOT NULL,

[Surname] [nvarchar](50) NULL,

[MiddleName] [nvarchar](50) NULL,

[FirstName] [nvarchar](50) NOT NULL,

[GenderCode] [nvarchar](50) NULL,

[CivilStatus] [nvarchar](50) NULL,

[DateOfBirth] [datetime] NULL,

[CountryCode] [nvarchar](50) NULL,

[StateCode] [nvarchar](50) NULL,

[LGACode] [nvarchar](50) NULL,

[PlaceOfBirth] [nvarchar](50) NULL,

[Height] [nvarchar](50) NULL,

[Weight] [nvarchar](50) NULL,

[HealthStatusCode] [nvarchar](50) NULL,

[ScreenCode] [nvarchar](50) NULL,

[PositionInFamily] [int] NULL,

[NoOfChildren] [int] NULL,

[CreatedOn] [datetime2](7) NULL,

[CreatedBy] [nvarchar](50) NULL,

[ModifiedOn] [datetime2](7) NULL,

[ModifiedBy] [nvarchar](50) NULL,

[Deleted] [bit] NULL,

[DeletedOn] [datetime2](7) NULL,

[DeletedBy] [nvarchar](50) NULL,

[TicketCode] [numeric](18, 0) NULL,

[PinCode] [numeric](18, 0) NULL,

CONSTRAINT [PK\_BioDatas] PRIMARY KEY CLUSTERED

(

[Code] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON) ON [PRIMARY]

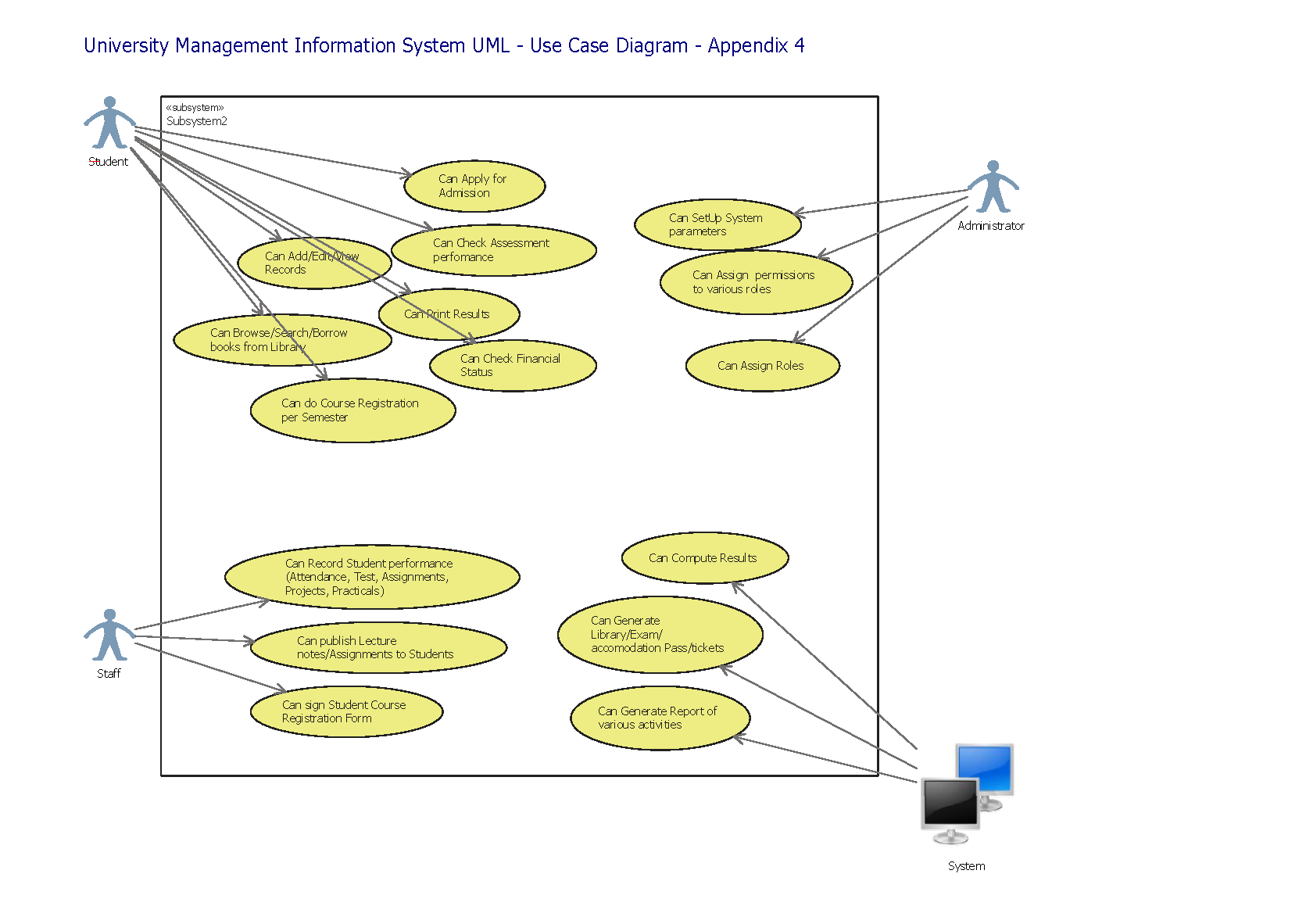
) ON [PRIMARY]

1. **Stored Procedures and Functions:** My tendency with stored procedure names is to explain what they do in plain English, using a verb-noun scheme. Prefixes that tell you right off the bat what the procedure is going to do for you, like [Personals].[SPBioDatasInsertUpdate]

**4.8 Use case diagram**

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases.

The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted. *(See diagram below for this project use case diagram or see appendix 4 for details)*



**4.9. User interface design**

User interface design or user interface engineering is the design of computers, appliances, machines, mobile communication devices, software applications, and websites with the focus on the user's experience and interaction. The goal of user interface design is to make the user's interaction as simple and efficient as possible, in terms of accomplishing user goals—what is often called user-centered design. Good user interface design facilitates finishing the task at hand without drawing unnecessary attention to it. Graphic design may be utilized to support its usability. The design process must balance technical functionality and visual elements (e.g., mental model) to create a system that is not only operational but also usable and adaptable to changing user needs.

Interface design is involved in a wide range of projects from computer systems, to cars, to commercial planes; all of these projects involve much of the same basic human interactions yet also require some unique skills and knowledge. As a result, designers tend to specialize in certain types of projects and have skills centered around their expertise, whether that be software design, user research, web design, or industrial design ( [*http://en.wikipedia.org/wiki/User\_interface\_design*](http://en.wikipedia.org/wiki/User_interface_design)).

**4.10. The Elements of Interface Design**

Interface design encompasses three distinct, but related constructs--usability, visualization, and functionality (Vertelney, Arent, & Lieberman, 1990). Recently, a fourth component of interface design has emerged as a critical factor--accessibility. Interface design is most often associated with the development of Web pages, computer software, and multimedia, but is relevant to the creation of any instructional media or technical equipment.

* Usability: refers to how intuitively or easily your media item is navigated and processed (flow, sequence, instructions, and download time). Note, this construct is the most inclusive of the three and is influenced by both visualization and functionality.
  + Planning your presentation (flow, navigation)
  + Navigation samples
  + About scaffolding
  + Heuristic evaluation
* Visualization: is creating visually interesting and aesthetically pleasing media items while avoiding potentially distracting or unnecessary "bells and whistles."
  + About visuals and icons
  + About text
  + Working with color
  + Screen design and layout
* Functionality: refers to the features of your media item and how useful they are for supporting a given task (e.g., interactive simulations, drill and practice quizzes, site maps, frequently asked questions, search engines).
* Accessibility: is an emerging Web interface design topic; if not addressed, it will negatively influence Web site usability for users with certain disabilities; tools that help users access your site in alternative formats (e.g., text, aural, visual) provide for increased functionality

**4.11. Web Interface Design**

How do your users get from point A to B?

Web interface design is a critical part of any website or web application. Good interface design can increase consumer confidence and increase productivity; however, for many programmers (and even for some designers), interface design is not part of their skill set.

If an organization uses an administration tool frequently, the interface design for that tool can be just as important as the front-end e-commerce website accessed by customers.

**4.12. What makes an intuitive interface?**

For an experienced interface designer, the nuances of interface design may seem obvious, but for many people interface design can seem intimidating. While interfaces can and do vary greatly, almost all intuitive interfaces include the following five features:

1. Global navigation
2. Limited navigation options
3. Back button
4. Home button
5. Location indicator

Simply integrating these five features into your interface can drastically improve its usability. But what do they really mean? Each of the five items is detailed below.

**4.13. Breaking it down**

**1. Global Navigation**

Some things are sacred in interface design, and a globally-consistent navigation is one of them. For websites, the global navigation may be major categories such as company, clients, design, etc., while for an e-commerce administration tool they might be orders, products, and advertising. Whatever the navigation elements, the navigation should stay readily available and in the same place throughout the entire interface.

**2. Limited Navigation Options**

Everybody likes choices, but in interface design too many choices can be bad. By limiting choices on a given page, you can help keep the user on track and help to avoid confusion. As a general rule, global navigation should have no more than ten options, and any subsets of navigation should also be limited to ten. If you find you can't keep the items limited to ten, then it may be a signal you need to re-categorize or utilize another type of navigation (e.g., a search box).

**3. Back Button**

Have you ever used an interface where you couldn't get back to the section you were just in? For many web applications, the problem of a lack of a back button is solved by the back button integrated into the browser itself. However, if your interface leads a user to a dead-end (for example, after a product update), you should always include a way to return to the previous page. When using Flash, it is important to keep in mind that unless you create separate HTML pages, the browser back button becomes completely useless, and in fact can be a little misleading. Users trying to return to the previous page may turn to the browser back button, and in doing so erase any memory of where they were in your Flash application!

**4. Home Button**

In the same vein as the back button, there should always be a one-step method for returning to the top or home page of an interface. A common practice is to associate the home page with the logo or header image of an interface. While there is nothing wrong with this, it is also good policy to actually have a global navigation item labeled home, index or start. This way you are providing not one but two methods for a user to return to the beginning of your application. This is beneficial as different users may have developed different habits, and in either case your interface will meet their expectations.

**5. Location Indicator**

If the home button is a quick way for a user to reset their compass, a location indicator is that compass. A location indicator gives the user an idea of where they are in an application, and is especially important in larger multi-tier applications. A location indicator can take a number of forms including navigation highlighting and breadcrumb navigation. Navigation highlighting is simply differentiating between the current section and page in the global and/or sub navigation on the page. Breadcrumb navigation is used most effectively on larger applications where navigation will proceed through several categories, allowing a user to back up to any point in the trail. For examples of breadcrumb navigation, look at the Yahoo! or dmoz directory.

**4.14. Testing your Interface**

After several hours (or days) of work, your interface has all of the five features mentioned above and you're ready to pop the cork on that bottle of champagne. Not so fast! Before declaring your interface a victory, you owe it yourself and your users to test it out. Find someone whose opinion you trust, but have had little or no exposure to the interface you just completed.

Ask them to use and comment on the interface you have created. While they are using the interface, pay attention to any details you need to point out to them, and take notes on any places they become stuck or unsure of how to precede — these may be problem areas that you can improve on. Pay careful attention not only to what they say about the interface, but what they do.

**4.15. When to break the rules**

The five key points mentioned above will help make any interface more intuitive and easy to use, but what if that is not what you are after? There are plenty of examples of non-commercial and even commercial interfaces that are intentionally non-intuitive. The purpose of these interfaces is to grab the user and make them work to figure out how to use it.

Based on the explanations and the model outlined above on interface design, the interface can be designed to cater for the needs of these specific users. And the Web interface for this project would be simple and intuitive. As mentioned in section 1.2 it is the overall aim of the end user to expand on the results of this project and implement a fully functional University Information Management System. It is therefore required that the design of this element will contribute to easing any future implementation by ‘keeping it simple’.

**4.16. Summary**

In this chapter we have discussed the system design which has taken place to ensure a structured implementation process will follow. We have examined both the database element and the interface element stages, outlining the important decisions made and the processes carried out to achieve the design guidelines required. In this chapter a main objective was met which meant the analysis data had been successful modeled into a design ready for implementation.

The data modeling was carried out to a point at which the data had a balance between the level of integrity and the level that could feasibly be implemented still allowing for a flexible system. As well as discussing the database design we also examined the interface design considering its usability, its layout and its structure.

The design processes and outcomes discussed in this chapter will be followed through into the next chapter, implementation.