Summary

The overall aim of the project was to develop a fully tested system that would automatically generate a set list for a DJ. The major challenge was to capture the knowledge of the DJ, and use this knowledge in generating a set list.

In tackling t he problem I met with the DJ, which resulted in a series of system requirements. These requirements stated exactly what was required by the system. After ensuring the proposed solution was feasible my next step was to research which system development me thodology would be the most appropriate to use in providing a solution. A DSDM methodology was selected on the grounds that my end user wanted to take an active part in the development of the system.

The design of the database followed a traditional data modelling approach. This involved developing an E-R model, and using this to map to a logical, fully normalised database design. The development of the system involved the use of Microsoft Access and VBA (Visual Basic for Applications).

It was identifi ed that there were two possible approaches to generating set lists. One approach used hard - rules; the other used a 'weightings' approach. Both approaches were implemented using VBA. The end user was heavily involved once implementation was complete. He was required to analyse both approaches, and select the approach that produced the best results. An in -depth discussion of the two approaches is included in the report.

After the best approach to generating a set list had been determined the system as a whole was fully tested. A user manual was provided to ensure the end user could effectively use the system.

Finally, the system was reviewed to see whether the system I built matched up with the initial requirements we set. The review also involved a ssessing whether the system would actually be of any real use to the end user in the future.

Acknowledgements

I would like to acknowledge the help of the following people in the completion of the project:

Stuart Roberts, my project supervisor, for the help and support he has given me throughout the course of the project.

Simon Peterson, my end user, for providing me with the opportunity to undertake this project. I am very grateful for the time and effort he has contributed.

Contents

Chapter 1 – Introduction

1.1 Aim of the Project

The overall project aim is to capture the knowledge of a DJ using IT, and use this knowledge to automate the process of generating a set list.

1.2 Overview of Current Problem

My end user is a part-time DJ. He has identified problems with his current way of working. At present searching for a specific track from his collection of records is done manually, rooting through his many boxes of records until he comes across the track he is looking for. This can prove time consuming. He is looking for a system that will allow him to search for any track in his entire collection, allowing him to identify where that track physically lies.

The major problem he is experiencing is preparing a set list before a gig. He currently drafts a rough list of tracks on paper, choosing the tracks almost randomly from those he can remember. He has found drawbacks with this method of preparation. Firstly, tracks are sometimes played after tracks of a completely different speed, leading to difficulties when trying to mix the two tracks together. Secondly, many good tracks from his collection are getting neglected, due to him not being able to either find them or remember them. Thirdly, the amount of tracks he selects doesn't usually fit in with the duration of the set. This leads to either filling the set with random tracks that don't fit in well with the style of the set, or having to leave out popular tracks if he runs out of time. To solve these problems a system is required that would generate a set list of tracks, using a series of rules that the DJ himself would use when deciding which tracks should be played after each other.

1.3 Project Objectives

I set the following objectives for my project:

1. Investigate system requirements

When developing any system it is extremely important to get a clear view of the problem, and obtain the exact requirements of the system to be put in place. I will need to conduct organised meetings with my end user to collect the required information.

2. Select tools for development of the system

Once the system requirements are clear the most suitable tools for developing the system can be selected. I need to ensure that the end user will have the software required to run any system I develop.

3. Cost the system

It is important to cost any system before deve lopment, to ensure that the proposed system is actually economically feasible.

4. Design and implement a system for storing tracks and generating a set list. A user manual should be included

This is the main task in the project: providing a system to sol ve the problems covered in section 1.2. To ensure that my end user, and any possible future users can effectively use the system I create, it is important that I develop a user manual.

5. Design and compare two different prioritising algorithms for use in set list generation

After discussion with my project supervisor we identified two different approaches to generating a set list. One uses 'hard-rules' to make track selections, the other 'soft -rules'. My objective is to implement the two different approaches, and use the end user to select the better of the two approaches. I will be discussing the relative merits of the two approaches.

1.4 Project Deliverables

Below is an outline of the deliverables for this project:

1. Working System

I identified a set of **minimum requirements** for system I was to develop. The system should:

a) Allow the details of records and the tracks from those records to be stored. This should provide for data input, deletion and update.
b) Search for specific tracks or records.
c) Generate a set list of duration given by the user.

A user-friendly interface should be developed for the system.

2. User Manual

To ensure that my end user, and any possible future users can effectively use the system I will be creating a user manual. This will explain the system in terms that non -technical users can understand, and will cover every aspect of the system.

3. Report

This report describes each stage that was completed during the development of the system, and details the processes and procedures that were used to successfully complete the project.

1.5 Selection of a System Development Methodology

A system development methodology is "a collection of procedures, techniques, tools, and documentation aids which will help the system developers in their efforts to implement a new IS. A methodology will consist of phases, themselves consisting of sub-phases, which will guide the systems developers in their choice of the techniques that might be appropriate at each stage of the project and also help them plan, manage, control and evaluate IS projects." *(Lau, L and McCormack, J, 2000)* I have researched a number of system development methodologies to decide which would be the most appropriate for conducting my project.

The **Waterfall Model** breaks down the development process into a series of stages, logically ordering the stages. The methodology requires you complete each stage in order: feasibility study, systems investigation, analysis, design, implementation, review and maintenance. This methodology has been criticised for being too rigid, making it difficult to respond to changing user needs, as users do not play an active part in the development. It is suggested that this methodology often leads to user dissatisfaction, as it doesn't give users the opportunity to see the system until is has been completed.

More recent methodologies attempt to address such problems. Rapid Application Development (RAD) "addresses the need to develop information systems quickly. It is not based upon the traditional life cycle, but adopts an evolutionary/prototyping approach." *(Avison, D and Fitzgerald, G, 1995)* A framework for RAD has been developed called **Dynamic Systems Development Method (DSDM)**. DSDM is based on nine principles. Some of these include:

§ Active user involvement is imperative.
§ DSDM teams must be empowered to make decisions.
§ Iterative and incremental development is necessary to converge on an accurate business solution.

(Lau, L and McCormack, J, 2000)

A major advantage of the DSDM approach is the flexibility it provides. Changes to user requirements can be responded to quickly, ensuring the end product matches what the user actually wants. As my end user wants to take an active part in the development of the system DSDM is the obvious choice. User involvement throughout the development process ensures users are not left with a system they don't want. The success of the system is also put in the hands of the user as well as the developer.

The 5 phases of DSDM are feasibility s tudy, business study, functional model iteration, system design and build iteration, and implementation. The following diagram illustrates these 5 phases:

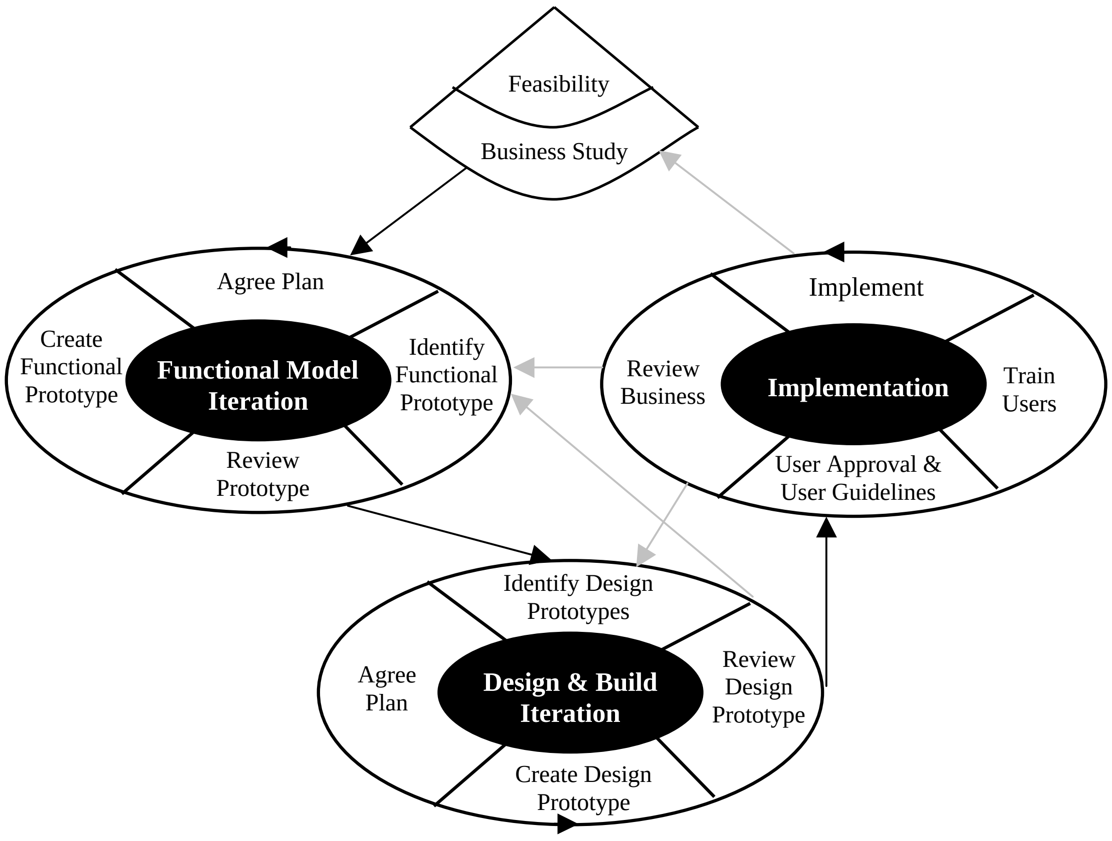


Fig 1.1

Fig 1.1 shows that DSDM   is an iterative approach: continual refinement of the system using the end user. My end user will be involved throughout the development of the system, and will be given the power to make decisions.

Chapter 2 – Analysis

2.1 System Requirements

In meeting with my end user I have obt   ained a series of requirements for the system, both functional and non-functional.

2.1.1 Functional Requirements

The functional requirements state what activities the system must perform:

1.1 The system shall record the details of every record. These de   tails include the record number, title, and label.
1.2 The system shall record the details of every track. These details include the record the track appears on, title, artist, mix, side, style, BPM (beats per minute), mood, popularity, whether the track can start a set, and whether the track can finish a set.
1.3 The system shall allow the details of a record or track to be updated.
1.4 The system shall allow records or individual tracks to be deleted from the system.
1.5 The system shall allow the details of any record or track to be retrieved.
1.6 The system shall generate a set list from the tracks in the system.
1.7 The system shall use selections made by the user to generate the set list. These selections include the duration of the set, and a starting style.
1.8 The system shall provide several alternative tracks to the tracks on the set list.
1.9 The system shall allow a track on the set list to be replaced by a track on the alternative list.
1.10 The system shall allow a track on the set list to be replaced by any track in the system.
1.11 The system shall provide the generated set list in a printable form.

2.1.2 Non-functional Requirements

The non-functional requirements cover usability and performance issues:

1.1 The system shall have a user-friendly interface, allowing users with only limited computing experience to effectively use the system.
1.2 The system shall include documentation on how to use the system.

2.2 Selection of Tools for Development

From studying the system requirements I decided that a Relational Database Management System (RDBMS) was the most suitable software to use. There were several RDBMS to consider, including MS Access, SQL Server and Oracle. Microsoft Access was the obvious choice to use, as firstly, my end user has it installed on his computer, and secondly, the database will be relatively small, so therefore the large -scale functionality that SQL Server and Oracle provide are not required. Visual Basic for Applications (VBA) is a 'cut -down' version of Visual Basic to use in conjunction with Microsoft Access. I will be using VBA to implement the functionality that cannot be achieved simply using Microsoft Access.

2.3 Existing Systems

I found a program on the Web called Song Librarian 2.0 (downloaded from http://www.hitsquad.com/smm/programs/The_Song_Librarian/). This program is similar to my proposed system in that it allows tracks to be added and searched for. The major flaws I found with this system are the options aren't presented to you in a logical order. F or example, to add an artist and style to a track you have to keep jumping to different places in the system, which is both time consuming and confusing. The interface is also cluttered, which again causes confusion. I will ensure that the interface I de velop does not suffer the same pitfalls.

2.4 Constraints

2.4.1 Time

To ensure that my system was implemented, tested and installed, along with a user manual within the time constraint set, I developed a project schedule by which to organise my time. This can be seen in Appendix B. This process of creating deadlines within the overall project deadline is known as *timeboxing*. It is a feature of the DSDM methodology I am using to develop my system.

2.4.2 Cost

The only costs I encountered were possible dat a input costs, as my end user has the required software (Microsoft Access) installed on his computer. I am planning on inputting roughly 50 records into the system, which will amount to roughly 150 tracks, for testing purposes. After estimation of input time, and cost to employ a data input clerk to input the remaining records, I arrived at a total of £45. The end user has informed me he is planning on inputting the remaining records himself. This means there will be no cost for the system.

Chapter 3 – System Design

3.1 Data Modelling

3.1.1 Overview

Data modelling is a technique for analysing the UoD (Universe of Discourse) and translating it into a graphical model, which can then in turn be translated to the tables in a relational database. (*Mott, P and Roberts, S, 1998)*

3.1.2 E-R Diagram

In E-R modelling everything in our UoD is modelled as an Entity, Relationship or Attribute. The E-R model is very important as it forms the foundation from which the database is built. If the E-R diagram is wrong, the resulting database will also be wrong. For my system I have identified the following entities: **Label**, **Record**, **Track**, **Neighbouring Track** and **Artist**. From this I have developed the following E-R diagram:

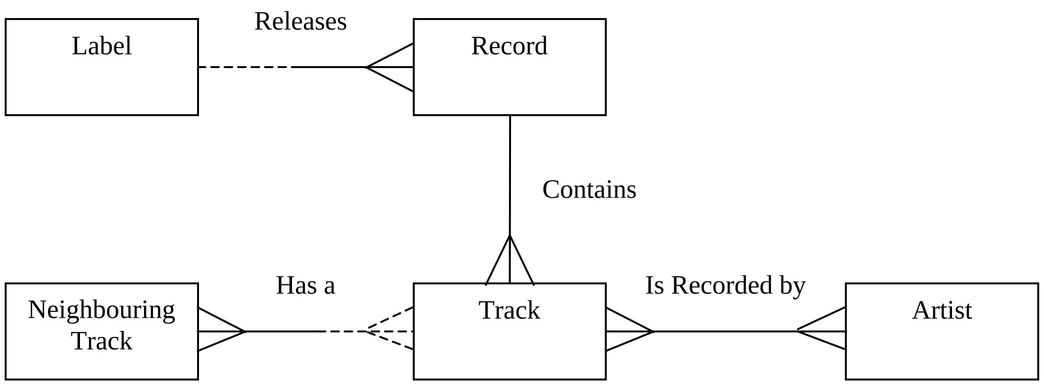


Fig 3.1

A **label** may release many records. A **record** may contain many **tracks**. A **track** may have one or more **neighbouring tracks** (a neighbouring track is a particular track that sounds good when played after that track). A **track** may be a **neighbouring track** of more than one track. Not every track will have a **neighbouring track**. More than one **artist** may appear on a **track**. An **artist** may appear on more than one **track**.

3.2 Logical Database Design

3.2.1 Mapping from E-R Model to Logical Database Design

To map from the above E-R model to a series of relational tables I have needed to follow a series of rules.

Firstly, each entity-type maps onto a table scheme, with associated attributes taken from those of the entity-type in the E-R model (primary keys are underlined):

Label(<u>Label_ID</u>, Label)
Record(<u>Record_Number</u>, Record_Title, Reuse)
Track(<u>Track_ID</u>, Track_Title, Mix, Side, Style, Speed, Popularity, Mood, Start_Set, Finish_Set, CurrentPlaylist, Classic, LastPlayed, LastStart, LastFinish)
Artist(<u>Artist_ID</u>, Artist)
NeighbouringTrack(<u>Track_ID</u>, <u>NeighbouringTrack_ID</u>)

In the 'Record' table the primary key is the *record number*. The record number is a unique number the end user sticks onto each record he buys. The higher the record number the newer the record.

For the *style* attribute in the 'Track' table the end user stated that each track in his collection could be categorised as one of 5 styles: Big Beat, Dark, Jump Up, Nu Skool and Techno.

The *speed* attribute is an estimation of the tracks bpm (beats per minute): **A**(<120 bpm), **B**(120-129 bpm), **C**(130-139 bpm), **D**(140-149 bpm), **E**(150+ bpm).

The *popularity* is a number between 1 and 9, reflecting how popular that particular track is.

The *mood* is a number between 1 and 5. It reflects the stage at which a track is best played during a set (0 = early stages of the set, 5 = late stages of the set).

A track on the *current playlist* is a track that you would regularly like to see appearing in the set lists. When a track is input into the system it is automatically defined as being on the current playlist. When a track starts to become outdated it can be taken off the current playlist, which means it will no longer appear on the set lists. If you would still like the track to appear once in a while it can be defined as a *classic* track.

For the **1:M relationships**, the primary key of the master table is posted in the related table.

i) **Label** releases **Record**

Label(Label_ID), **Record**(*Label_ID*)

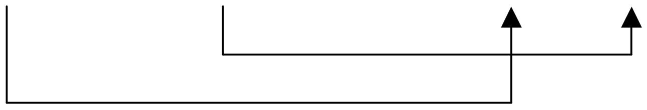
ii) **Record** contains **Track**

Record(Record_Number), **Track**(*Record_Number*)

The **M:N relationship-types** map onto relationship schemes. The attributes of the relationship schemes are the two primary keys of the entity-types participating in the relationship:

i) **Track** is recorded by **Artist**

Track(Track_ID), **Artist**(Artist_ID), **Track Artist**(Track_ID, Artist_ID)



ii) **Track** has a **Neighbouring Track**

Track(Track_ID), **NeighbouringTrack**(Track_ID, NeighbouringTrack_ID)

The 'NeighbouringTrack' table is a pairing of two tracks The *Track_ID* is the *Track_ID* of the first track in the neighbouring relationship, *NeighbouringTrack_ID* the *Track_ID* of the second track. Both fields are selected as the primary key. This means that each neighbouring relationship must be unique.

The resulting database schema was developed (primary keys are underlined, foreign keys are italicised):

Label(Label_ID, Label)

Record(Record_Number, *Label_ID*, Record_Title, Reuse)

Track(Track_ID, *Record_Number*, Track_Title, Mix, Side, Style, Speed, Popularity, Mood, Start_Set, Finish_Set, CurrentPlaylist, Classic, LastPlayed, LastStart, LastFinish)

Artist(Artist_ID, Artist)

Track Artist(Artist_ID, Track_ID)

NeighbouringTrack(Track_ID, NeighbouringTrack_ID)

The data dictionary can be found in APPENDIX C.

3.3 Normalisation

Normalisation is an essentia l part of database design. Elmasri, R and Navathe, S (1999) state that the purpose of normalisation is to (1) minimise data redundancy, and (2) minimise insertion, deletion, and update anomalies. Data redundancy is said to occur when the same piece of da ta is held in more than one place in a database, which can cause inconsistencies to occur. Insertion, deletion, and update anomalies can lead to serious maintenance problems.

Functional Dependencies

For any relation, R, the set of attributes, {A_1, …A_n} is said to **functionally determine** an attribute B of R if any two rows of R that have the same values for {A_1, …A_n} MUST have the same value for B. *(Roberts, S, 1999)*

The following functional dependencies exist in my database:

Label

Label_ID à Label

The *Label_ID* is used to uniquely determine each label.

Record

Record_Number à Record_Title, Label_ID, Reuse

The r*ecord number* is unique for each record, so therefore allows all attributes of the record to be identified. For my model I am assuming that more than one record may have the same title. This means that from knowing the title of a record you cannot determine the record number.

Track

Track_ID à Record_Number, Track_Title, Mix, Side, Style, Speed, Popularity, Mood, Start_Set, Finish_Set, CurrentPlaylist, Classic, LastPlayed, LastStart, LastFinish

The *Track_ID* is used to uniquely determine each track in the system. The *speed* and *mood* attributes are independent of each other, so from knowing the speed of a track you cannot determine the mood.

Track Artist

Track_ID à Artist_ID

Artist

Artist_ID à Artist

The *Artist_ID* is used to uniquely determine each artist.

NeighbouringTrack

No functional dependencies exist in the **NeighbouringTrack** relation as both attributes make up the primary key:

NeighbouringTrack{Track_ID, NeighbouringTrack_ID}

Knowing a *Track_ID* does not necessarily give you the *NeighbouringTrack_ID*, as a track may have more than one neighbouring track. Also, knowing a *NeighbouringTrack_ID* does not necessarily give you the *Track_ID*, as a track may be a neighbouring track of more than one track.

For a relation to be in **First Normal Form (1NF)** no field may accept multi-valued entries. No field in my database accepts multi-valued entries. Therefore 1NF is upheld.

A relation is in **Boyce Codd Normal Form (BCNF)** whenever there is a non-trivial dependency, $A_1, \ldots A_n$ à B for R, it is the case that $\{A_1, \ldots A_n\}$ is a superkey for R.
(Roberts, S, 1999)

All the non-trivial dependencies I have displayed from my database conform to BCNF. In all relations a single attribute determines all other attributes in the relation. Therefore I can conclude that my database is fully normalised in BCNF.

3.4 Integrity Constraints

It is important to ensure that primary keys are properly and fully defined. The following two rules help to ensure that primary and foreign keys are properly defined for each entity:

Entity Integrity: No attribute participating in the primary key of a base table is allowed to accept null values. *(Mott, P and Roberts, S, 1998)*

Every primary key in the database has an auto-number data type, except *record number,* which is the primary key in the record table. This means that whenever a new record is created, those auto-number fields are automatically assigned a number, ensuring that they are never null. Validation checks will be made to ensure the user can never leave the *record number* field blank when inputting a new record.

Referential Integrity: The foreign key of a table must be either equal to the value of the primary key in the matching table, or be wholly null. *(Mott, P and Roberts, S, 1998)*

By defining relationships in Microsoft Access referential integrity can be upheld. This is discussed in depth in section 4.1.2.

3.5 Human Computer Interaction

3.5.1 Overview

Human Computer Interaction (HCI) is defined as "the study of the interaction between people, computers and tasks. It is principally concerned with understanding how people and computers can interactively carry out tasks". *(Johnson, P, 1992)* This is very relevant to my project, as my end user only has limited computing experience. The user interface I develop will determine how successful the final system will be. I could develop an excellent underlying system, but if my end user has difficulty operating the system, the system cannot be deemed a success.

It is important to consider any possible future users of the system when developing the user interface. It is likely that any possible future users of my system will only have limited computing experience, like my end user. I need to ensure that my system is usable by any computer novice in general, not just my end user.

3.5.2 Interface Design

The main purpose of a user interface is to hide the complexities of the underlying system, and make navigation through the system as convenient as possible. As far as the user is concerned the interface is the system. When designing the user interface a number of considerations must be taken into account:

Consistency
Consistency is a key factor in developing a successful user interface. It helps the user transfer familiar skills to new situations, and increases the speed at which a user can get to grips with an interface. Consistency will be achieved by maintaining the same layout and colour scheme, and ensuring the operation of the interface is consistent throughout the interface.

Colour
Colours should be used that are easy to view and not harsh on the eyes. I have decided on a simple colour scheme: a grey background with mainly black text.

Colour should be used in a way that is consistent with the expectations of your user. For example red is a colour that is associated with warning. I will be using red to issue my user warnings. The simple grey background will attract the user to such warnings when they appear on-screen.

Usability

The interface should aim to maximise the speed and ease at which operations can be carried out. The tab key will be used to enable the user to scroll through the functions in a logical order. This will restrict the mouse usage, which will increase the speed at which the user will be able to operate the system.

Horizontal scroll bars are inconvenient for users. Where possible I will try to present the information so that the horizontal scroll bars are not required.

The ID fields that are solely used as primary keys for the tables in the database should not be displayed on the user interface. The fields have no relevance to the users of the system, so should not be displayed. Including ID fields on the interface is likely to confuse the user of the system.

Error Messages

Error messages are usually overlooked as an important part of interface design. Well-designed error messages help to prevent the user getting frustrated with the system, and can aid learning of the system. I feel that error messages should be split into two groups, and treated differently:

1) Error messages that appear as a result of the user making a mistake, for example, trying to save a record number that already exists. In these cases the error message should inform the user what they did wrong, without been insulting, for example: "The record number you selected is already in use. Please enter a unique record number."

2) Error messages to inform the user that the system could not fulfil a valid operation, for example, the system not being able to find a suitable starting track for a set list. In these cases the error message should apologise to the user, so they know it wasn't something they did wrong, for example: "I'm sorry. There was no suitable starting track."

Interface Development

After initially sketching a few ideas for the interface, I developed a series of prototypes using Microsoft Access that displayed how I thought the interface should look. Once the end user was happy with the interface I began implementing the functionality.

The system interface can be seen in Appendix E.

Interface Navigation

After grouping together similar functions, I decided on the following hierarchical structure for the interface:

1. Input

1.1 Add New Record

- 1.1.1 Add Record
- 1.1.2 Add Track
- 1.1.3 Add Artist

1.2 Add Track to a Record

- 1.2.1 Find/Delete Record
- 1.2.2 Add Track
- 1.2.3 Add Artist

2. Search

2.1 Find Record

- 2.1.1 Find/Delete Record
- 2.1.2 Display Tracks
- 2.1.3 Update Track

2.2 Find Track

- 2.2.1 Find/Delete Track
- 2.2.2 Update Track

3. Generate Set

3.1 Generate Set List

4. Neighbouring Tracks

4.1 View Neighbouring Relationships

- 4.1.1 View the Tracks having Neighbouring Tracks
- 4.1.2 View/Delete the Neighbouring Relationship

Chapter 4 – Implementation

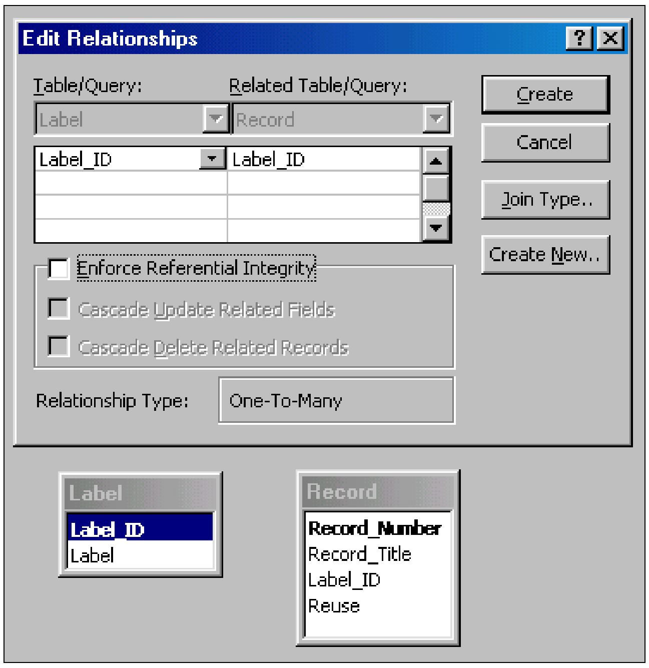
4.1 Database Implementation

4.1.1 Schema Implementation

Each table was implemented using the table builder in Microsoft Access, following the schema decided in the design stage (see section 3.2.1). Each table required a **field name** and **data type**. Data type exampl es include *text*, *auto-number*, *number*, *yes/no*. After entering in the fields Microsoft Access made it easy to define the primary keys I had previously designed: simply highlighting the specific field and clicking on the primary key icon.

4.1.2 Relationship Building

After creating the tables my next step was to define the relationships between the tables I had previously designed. Correctly defining relationships ensures **referential integrity** is upheld (see section 3.4 for a definition). Microsoft Access p rovides an excellent utility for defining relationships. Each table is displayed, and a relationship is defined between two tables by clicking on the primary key of one table, and dragging this field to the foreign key of the related table.



Table/Query:	Related Table/Query:
Label	Record
Label_ID	Label_ID

☐ Enforce Referential Integrity
☐ Cascade Update Related Fields
☐ Cascade Delete Related Records

Relationship Type: One-To-Many

Label
- Label_ID
- Label

Record
- Record_Number
- Record_Title
- Label_ID
- Reuse

Fig 4.1

Fig 4.1 shows the form that initially appeared after clicking on the primary key (*Label_ID*) in the Label table, and dragging this field to the foreign key (*Label_ID*) in the Record table. Microsoft Access identifies the relationship as a 1:M relationship. This is consistent with my E-R model (section 3.1.2).

As part of the referential integrity checks there are two options:

1) **Cascade Update Related Fields**: Any time you change the primary key of a record in the primary table, Microsoft Access automatically updates the primary key to the new value in all related records. This is not relevant to those tables with auto-number primary keys, as auto-number primary keys cannot be changed. The only table in the database not having an auto-number primary key is the 'Record' table. '*Cascade update related records'* is set for the relationship between the 'Record' table and the 'Track' table. This ensures that if the r*ecord number* changes, the record number stored for the tracks on the record will also be updated.

2) **Cascade Delete Related Records**: Any time you delete records in the primary table, Microsoft Access automatically deletes related records in the related table. This is set for the relationship between the 'Record' table and the 'Track' table to ensure that if a record is deleted, the tracks from the record are also deleted. 'Cascade delete related records' is also set for the relationship between the 'Track' table and the 'Track Artist' table, as this ensures that if a track is deleted from the system the artist is removed from the track. I did not set 'cascade delete related records' for the relationship between the 'Label' table and the 'Record' table, because if a label is deleted, the records produced by that record still exist, so therefore shouldn't be deleted.

For the 'NeighbouringTrack' table I needed to define two relationships, one between *Track_ID* in the 'Track' table and *Track_ID* in the 'NeighbouringTrack' table, and one between *Track_ID* in the 'Track' table and *NeighbouringTrack_ID* in the 'NeighbouringTrack' table. 'Cascade delete related records' is set for both these relationships to ensure that if either track from a neighbouring relationship is deleted, the neighbouring relationship is also deleted.

A graphical representation of the relationships can be seen in Appendix D.

4.1.3 SQL Queries

SQL (Structured Query Language) is a relational data language. I have used SQL extensively in the development of my system. Firstly, I have used SQL **INSERT** statements for tasks such as saving details to the database, and inserting tracks into the set list. The following example shows the SQL INSERT statement used to save the details of a new record to the system:

```
DoCmd.RunSQL ("INSERT INTO Record(Record_Number, Record_Title, Label_ID, Reuse)
VALUES(Record_Number, Title, Label, Reuse);")
```

The `VALUES` are the text box values entered by the user.

Secondly, SQL **UPDATE** statements are used for tasks such as updating the last played date of tracks, and updating track details. The fo llowing example shows the SQL UPDATE statement used to update the last played date of tracks on a set list:

```
DoCmd.RunSQL ("UPDATE Track SET LastPlayed = GigDate WHERE Track_ID =
forms!setList.Track_ID;")
```

The `GigDate` is the date entered by the user.

Thirdly, SQL **DELETE** statements are used for tasks such as deleting records and tracks from the system, and for emptying the set list. The following example shows the SQL DELETE statement used to delete a track from the system:

```
DoCmd.RunSQL ("DELETE FROM Track WHERE Track_ID = TrackID.value;")
```

SQL **SELECT** statements are used throughout the system to retrieve specific data. Examples from the system include:

a) Find the highest record number currently in the system. This is required for displaying the value t o the user on the ‘Add Record’ form.

```
SELECT Max(Record.Record_Number) + 1 AS MaxOfRecord_Number FROM Record;
```

b) Find all the classic tracks that haven’t been played in the last two weeks, and aren’t currently on the set list:

```
SELECT Record.Record_Number, Record.Record_Title, Record.Reuse, Track.Track_ID,
       Track.Track_Title, Track.Mix, Track.Style, Track.Speed, Track.Mood,
       Track.CurrentPlaylist, Track.Popularity, Track.Selected,
       Track.LastPlayed, Track.Classic
FROM  ((Label INNER JOIN Record ON Label.Label_ID = Record.Label_ID)
       INNER JOIN Track ON Record.Record_Number = Track.Record_Number)
       INNER JOIN (Artist INNER JOIN [Track Artist] ON
       Artist.Artist_ID = [Track Artist].Artist_ID) ON Track.Track_ID = [Track
       Artist].Track_ID
WHERE  (((Track.Selected)=No) AND ((Track.Classic)=Yes) AND ((Date()-
       [Track].[LastPlayed])>14))
ORDER BY Track.Popularity DESC, Record.Record_Number DESC, Track.LastPlayed;
```

c) Display all distinct track titles in the system. This is used as a control source for the tr ack title combo-box on the search form.

```
SELECT DISTINCT Track.Track_Title FROM Track ORDER BY Track.Track_Title;
```

4.1.4 Reports

Functional requirement 1.11 states that *the system shall provide the generated set list in a printable form.* To achieve this I created a report to display the set list in a clear format. This report can be seen in Appendix H.

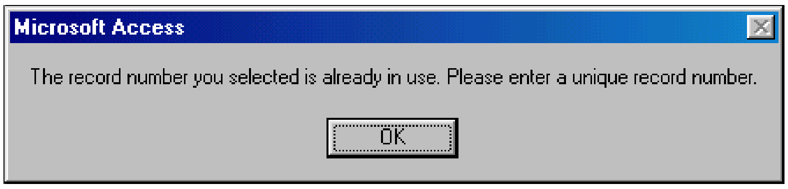
4.1.5 Input Validation

It is essential that users cannot input invalid data into the system. Validation mechanisms have been put in place to ensure that the user cannot:

a) **Input an incorrect data type** . For example, if you attempt to enter *text* into the 'Record Number' field on the 'Add Record' form an error message appears informing the user they have entered an incorrect data type. This is achieved by setting the *Format* attribute on the properties of the text box.

b) **Leave fields null that require a value** . For example, failing to enter a 'Record Title' on the 'Add Record' form produces an error message. This is achieved using VBA. Before the code e xecutes the SQL INSERT statement, checks are made to see whether any fields have been left null.

c) **Enter a non-unique value for a primary key**. For example, attempting to enter a 'Record Number' that already exists produces an error message (Fig 4.2). This is achieved by creating a *recordset* of all the records currently in the system using VBA, and searching through the recordset to see whether the record number selected by the user already exists. If so the error message is called.



The record number you selected is already in use. Please enter a unique record number.

Fig 4.2

4.1.6 Compaction

Generating a set list involves inserting tracks into a temporary table, and then deleting the tracks from this table before generating a new set list. As tracks are deleted from the temporary table on a regular basis, the database becomes fragmented and uses disk space inefficiently. Compacting a database rearranges how the file is stored on disk, minimising the fragmentation, and increasing the performance. I will be giving instructions on how to compact the database in the user manual.

4.2 Interface Implementation

Microsoft Access provides an excellent facility for developing user -friendly interfaces. Starting with blank forms I built the interface using controls provided by Microsoft Access. These controls include **text boxes**, **combo-boxes**, **check boxes**, and **command buttons**. *Text boxes* are used to obtain data input. *Combo-boxes* provide a way to display the user with possible values for an attribute, requiring the user to select a value from the combo box. This is an exc ellent form of data validation. For example, a combo -box is used to obtain a *style* value by providing a list of the five possible styles. This ensures that the value obtained is valid, as only a *style* from the combo-box is accepted (Fig 4.3).

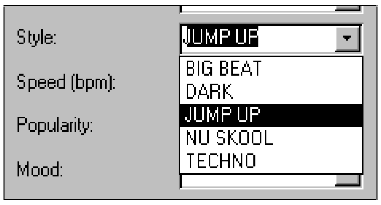


Fig 4.3

Check boxes are used on several occasions to obtain a yes/no response. For example, when inputting a track, check boxes are used to obtain whether that track is a suitable starting or finishing track. If so, the relevant box is simply checked (Fig 4.4).

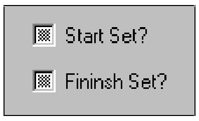


Fig 4.4

Command buttons are used on every form, providing the user with an excellent way to navigate through the interface. VBA code is executed when a button is clicked, performing tasks such as opening forms.

4.3 Set List Generation

The basis for generating a set list involves prioritising each track in the system (explained in section 5), and then inserting the most appropriate track into the set list. The set list obviously needs to be stored somewhere. I decided to create a temporary table, where the tracks could be inserted and stored until another set list was generated. This table was named 'setTemp'.

To generate a set list the user is required to enter the *duration* and a *starting style*. The most appropriate starting track is selected and inserted into the set list, and then the tracks are prioritised, with the most appropriate track being selected and inserted, and so on until the duration is filled. The most appropriate finishing track is then selected and inserted into the set list. My end user estimated that each track would be played for roughly 4 minutes. The prioritising and inserting is executed within a while loop, which is repeated until the required duration is achieved. To insert a track into the set list an SQL INSERT INTO statement is used, inserting the track details into the temporary table 'setTemp'. The following is a simplified example, showing how the Track_ID, and Track_Title of the most appropriate track would be inserted into the table *setTemp*:

```
DoCmd.RunSQL("INSERT INTO setTemp(Track_ID, Track_Title) VALUES(Track_ID,
Track_Title);")
```

4.4 Manipulating the Set List

After writing the code for generating a set list I came across the problem of being able to manipulate the set list. Specifically my end user identified he would need to be able to:

i) Add tracks to the set list.
To insert a track into the set list the user selects the track from a search screen, and then selects the position they want the track moving to by clicking on the insert button next to that position. Firstly, the position numbers are updated. For example, if you currently have 6 tracks on the set list and you want to insert **track X** at position 3, the tracks with position numbers 3 and above i.e. 3, 4, 5, and 6, all have their position numbers increased by 1. **Track X** is then inserted into *setTemp* using an SQL INSERT statement, and allocated position number 3. New records are always inserted at the end of the table so the position numbers would now appear 1, 2, 4, 5, 6, 7, **3**. When displaying the set list i.e. *setTemp*, to the user, the table is ordered by the position number. This displays the tracks in the correct order (1, 2, **3**, 4, 5, 6, 7).

ii) Change a tracks position on the set list.
To change a tracks position the user double-clicks on the position of that track. The user then selects the position they want the track moving to by clicking on the insert button next that position. Firstly, the positions are updated. For example, if you have 6 tracks on the set list and you want to move **track Y** from

position **2** to position **4**, the tracks with position numbers greater than 2 all have their position numbers *decreased* by 1. The positions now appear 1, **2**, 2, 3, 4, 5, 6. **Track Y** is then deleted from *setTemp* using an SQL DELETE statement. This leaves the order 1, 2, 3, 4, 5. We now need to update the positions again. The tracks with position numbers 4 and above i.e. 4 and 5, have their position numbers increased by 1, giving 1, 2, 3, 5, 6. **Track Y** is then inserted into *setTemp* using an SQL INSERT statement, and allocated position number 4, giving 1, 2, 3, 5, 6, **4**. As I mentioned previously, the set list i.e. *setTemp*, is ordered by the position number. This displays the tracks in the correct order (1, 2, 3, **4**, 5, 6).

iii) Delete tracks from the set list.

This involves deleting the track from the system using an SQL DELETE statement, and then decreasing the position numbers of those tracks with greater position numbers.

4.5 Neighbouring Tracks

4.5.1 Defining Neighbouring Tracks

Neighbouring tracks are defined as two tracks that sound good when played together, in the form *'track X should follow track Y in a set'*. Full consideration was put into the best way a user could define neighbouring tracks. I decided that neighbouring tracks should be able to be defined on the form that displays the generated set lists. Using VBA I implemented this functionality as follows: The user double-clicks on the *Track Title* of the first track in the neighbouring relationship. The *Track_ID* of this track is sent to a hidden text box on the form. The user then double-clicks on the *Track Title* of the second track in the neighbouring relationship. The *Track_ID* of this track is sent to another hidden text box on the form. The user is then asked to confirm that they want the neighbouring relationship saving, and then the Track_ID of each track is inserted in the 'NeighbouringTrack' table:

```
DoCmd.RunSQL ("INSERT INTO NeighbouringTrack(Track_ID, NeighbouringTrack_ID)
VALUES(Track1.Value, Track2.value);")
```

Validation checks are in place to ensure that you cannot define a track as a neighbouring track of itself.

Neighbouring tracks can also be defined on the 'Find Track' form, allowing the user to define any two tracks in the entire system as neighbouring tracks.

The neighbouring tracks functionality helps to capture the knowledge of the DJ. When a neighbouring track is inserted into the set list the constraints are overridden, because defining a neighbouring track is in-effect like saying "regardless of the two tracks style, speed, mood etc. it works well when the two tracks are played together".

4.5.2 Displaying Neighbouring Tracks

If a track on the set list has a neighbouring track that is not on the set list, the neighbouring track is displayed on the form, along with the position of the track it neighbours. The user can then simply double -click on the position of the track it neighbours, and it will be automatically inserted into the set list in the correct position.

4.6 Giving Variety to the Set Lists

The end user has emphasised that he doesn't want a tracks neighbouring track to be played in every set list generated, as this wouldn't provide the variety he is looking for. He would never play exactly the same s et week after week.

To provide this variety I have used random number generation, whereby the code generates a random number between 1 and 20, and executes a certain section of code depending on the number that is generated:

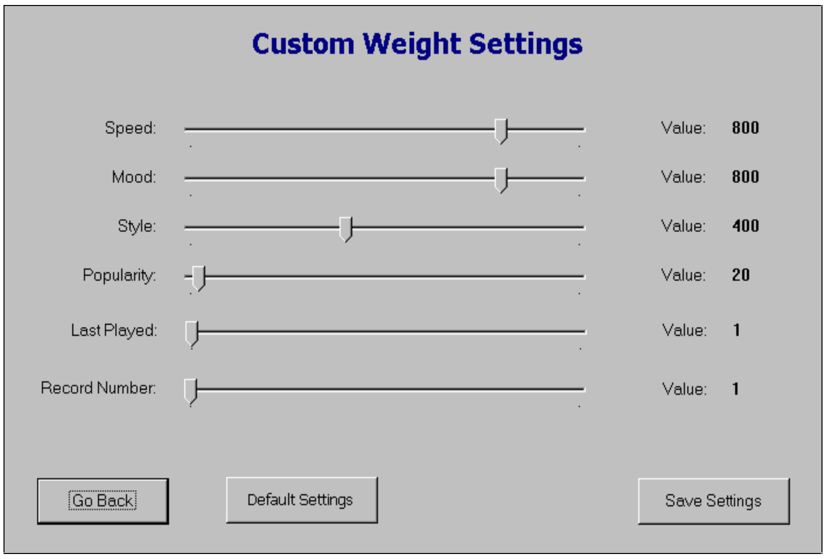
A) *Between 1 and 10*: Insert the most appropriate track from the current playlist.
B) *Between 11 and 19* : Insert a neighbouring track of the previous track, if the previous track has a neighbouring track.
C) *20*: Insert the most appropriate classic track.

My end user has expressed t hat he wouldn't play a classic track early on in a set. The code never selects a classic track within the first 30 minutes.

4.7 Customised Weight Settings

The second approach to generating a set list (discussed in detail in chapter 5) involved prioritisi ng tracks by multiplying attributes with carefully chosen parameters. Fig 5.2 shows a section of the SQL query that was used to prioritise the tracks. The parameters are shown in bold. The values of these parameters are very important, as they manipulat e how much weight is applied to each constraint. I spent a good deal of time with my end user fine -tuning these parameter values so that they generated the best possible set lists. We identified that the parameter values we had arrived upon should be use d as default each time, but the user should also be able to alter the parameter values as required, thus providing the flexibility to define different types of set each time.

My design decision was to implement a form that allowed the user to easily and ac curately adjust these parameters. I decided to implement the form as a control panel, with a series of sliding bars for each constraint (Fig 4.5). This form works well because it displays visually how much weight each constraint carries. I could have simply implemented this form using a series a text boxes, allowing to type in values for each constraint, but that wouldn't have been as visually effective.



Custom Weight Settings

Speed:	Value: 800
Mood:	Value: 800
Style:	Value: 400
Popularity:	Value: 20
Last Played:	Value: 1
Record Number:	Value: 1

Fig 4.5

The sliding bars are ActiveX controls (Microsoft Slider Control, Versio n 5.0). When the user moves a sliding bar and clicks on the *Save Settings* button, the value of the text box to the right of the sliding bar is updated to reflect the new position of the sliding bar. The values from these text boxes are used in the code when set list is generated. The following code example shows how the *mood* value from form Fig 4.5 would be used:

```
moodParam = Forms!weightSettings!moodSliderValue.Value
```

This would assign the variable `moodParam` a value of 800.

The *Default Settings* button resets the text box values and sliding bars to the values we decided upon.

4.8 Evolution of User Requirements

As the DSDM methodology I was using emphasises user involvement, the user requirements changed on a regular basis as a result of the meetings I had with my end user. Several changes were made to the system after I presented it to the end user. Firstly, he identified that the system allowed several tracks from the same record to be included on the same set list. This is not what was wanted. He informed me that he would play more than one track from only a few of the records in his collection on the same set list, and he wanted this capturing by the system. We decided upon adding an attribute to the 'Record' table called *Re-use*. This attribute was given a yes/no data type. *Re-use* is set to yes when inputting a record, if the record contains tracks that could appear on the same set list. If more than one track from a re-useable record is selected on the same set list, the system notifies the user (Fig 4.6).

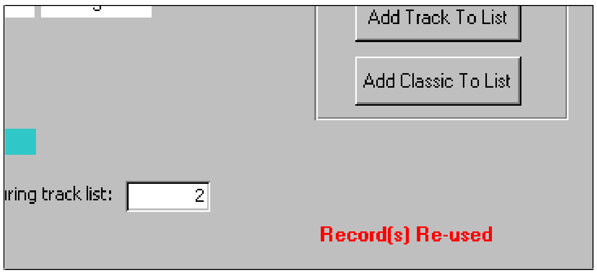


Fig 4.6

Checking whether any records are re-used involves taking a count of the number of tracks and distinct record numbers on the set list. If the number of distinct record numbers is less than the number of tracks it shows that a record has been re-used.

Each track had a *mood* attribute attached to it, which was a value between 1 and 5. This was an attribute suggested by the end user, defined as how 'heavy' the track was. The rules for generating a set list currently ignored the mood constraint. This meant that the resulting set lists generated had the mood jumping around completely randomly, for example, a track with mood 1 would sometimes follow a track with mood 5. My end user decided that he wasn't happy with the current definition of the mood attribute. Collectively we re-defined mood. The *mood* now reflects where in a set a track is most suited to appear. For example, tracks that are most suited to appearing at the start of a set are assigned mood 1, those that would be played in the middle of the set are assigned mood 3, and those that would be played near the end of a set are assigned mood 5.

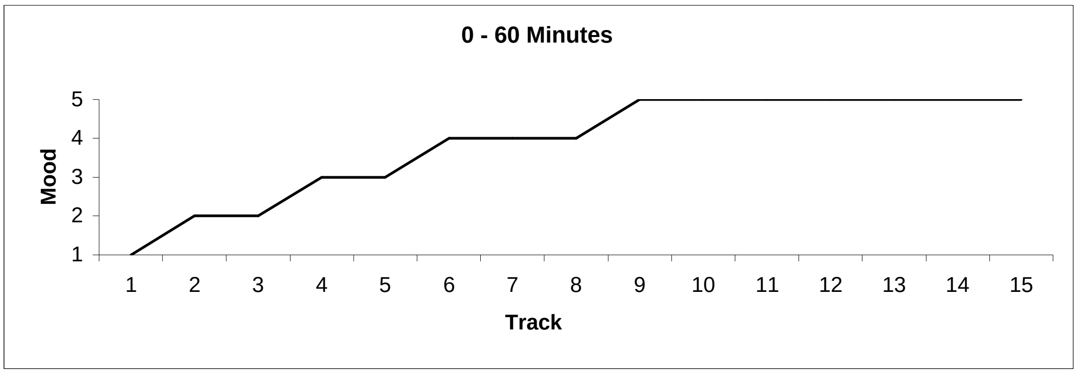
After mood was re-defined my end user wanted a mood constraint to be applied to the process of generating a set list. We came up with the idea of developing mood templates for different types of set. These mood templates would control the mood at each stage of the set. We decided upon the following four **set types**:

Background: The mood randomly fluctuates between 1 and 5.
Bangin': The mood starts at 1 and gradually builds up to 5. The speed gradually increases.
Chilled: The mood stays between 1 and 2.
Club: The mood randomly fluctuates between 2 and 4.

The most challenging set type to implement was the **Bangin'** set. This is the most common set my end user plays. In this type of set the mood to starts off at 1 and gradually builds up to 5 near the end of the set. He sketched me a graph of mood against time to illustrate this. The added complication with this template is the graph needs to be altered for different length sets so that the set always peaks at the correct time. Working with my end user we developed accurate graphs for 4 different length sets: 60 -minute, 120 -minute, 180 -minute, 240-minute.

Fig 4.7 shows the graphs for a) 60 -minute set, b) 240 -minute set. The x -axis interprets set duration as track numbers, assuming each track is played for 4 minutes.

a)



b)

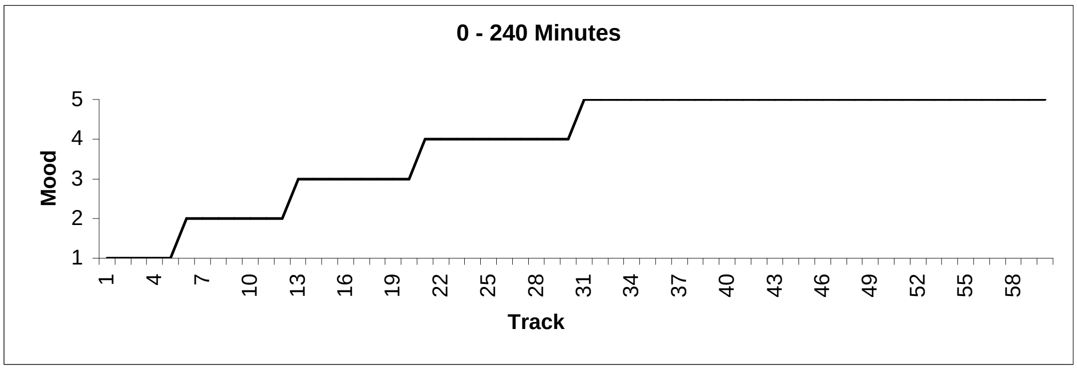


Fig 4.7

I converted each of t he 4 graphs into 4 arrays. Each value in the array reflects the mood that number track should be i.e. track *n* should have a mood corresponding to the *nth* entry in the array. These arrays are displayed in Fig 4.8. *DurationSelect* is the duration the user specifies. If you look at the first array on line 2 i.e. the array that will be used if the duration is less than 60 minutes, you will see that the first track in the set needs to have mood **1**, the second track **2**, third track **2**, fourth **3**, and so on.

```
If DurationSelect <= 60 Then
    BanginArray = Array(1, 2, 2, 3, 3, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5)

ElseIf DurationSelect > 60 And DurationSelect <= 120 Then
    BanginArray = Array(1, 1, 2, 2, 2, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5,
5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5)

ElseIf DurationSelect > 120 And DurationSelect <= 180 Then
    BanginArray = Array(1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4,
4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5)

ElseIf DurationSelect > 180 Then
    BanginArray = Array(1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3,
3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5)
End If
```

Fig 4.8

The array is selected each ti me according to the length of the set (as defined by *DurationSelect*). This ensures that the set always peaks at the correct point.

Chapter 5 – Comparison of Two Prioritising Algorithms

5.1 Overview

The process of generating a set list involves selecting a track, and then assessing all the other tracks in the system to find the most appropriate track to be played next. This process obviously requires a mechanism for prioritising the tracks in the system. I identified two possible approaches to doing this, a hard-rule approach, and a soft-rule "weightings" approach. My task was to implement both approaches, and then assess which approach produced the best results.

5.1.1 Hard-Rule Approach

This approach to generating a set list involves applying constraints to all tracks in the system. The purpose of these constraints is to find all the tracks in the system that are suitable to be played after the most recently selected track. Those tracks that are deemed suitable are then prioritised to find the single most appropriate track. This track is then inserted into the set list, and the process repeated: the constraints are applied using the newly inserted track, and the remaining tracks are prioritised. This approach is *hard-ruled* as those tracks that don't pass the constraints are completely ruled out of being played.

Constraints

The following constraints were developed with help from the end user:

Style Constraint: In the system every track is allocated one of five possible styles: Big Beat, Dark, Jump Up, Nu Skool, or Techno. The style constraint says which styles can follow which other styles. For example, if a track has been selected with a *Dark* style, the next track can only have a *Dark*, *Techno*, or *Nu Skool* style. Tracks not having these styles i.e. those having a *Big Beat* or *Jump Up* style are completely rejected. This constraint is applied to ensure that two tracks having styles that don't mix well together are not selected together in a set list.

Mood Constraint: Section 4.8 explained how 4 set types were identified: Background, Bangin', Chilled, and Club. These four set types state what kind of mood constraint should be applied. For example, the Chilled set type says that the mood must stay between 1 and 2. When using a Chilled set type, every track that does not have a mood value of 1 or 2 is automatically rejected. The Bangin' set type says exactly what mood each track should be. This is captured in a series of arrays. For example, if the user has selected to play a 60-minute set, and the first 5 tracks had been selected by the system, the next track (track 6) must have mood a mood value 4 (this is taken from the first array in Fig 4.8). Tracks without mood 4 are completely rejected regardless of their style and speed.

Speed Constraint: In the system every track is allocated one of five possible speeds. **A**(<120 bpm), **B**(120-129 bpm), **C**(130-139 bpm), **D**(140-149bpm), **E**(150+ bpm). The speed constraint says which speeds can follow which other speeds. The speed constraint th at is used depends on the set type that has been selected. For a Bangin' set the speed must gradually increase. For example, if a track has been selected with a *B* speed, the next track can only have a *B*, or *C* speed. Tracks without these speeds are compl etely rejected. For the other three set types the speed can also decrease. For example, if a track has been selected with a *B* speed, the next track can have an *A*, *B*, or *C* speed.

The complete constraints can be seen in Appendix F.

Prioritisation

After the constraints have been applied, the remaining appropriate tracks are prioritised to leave the single most appropriate track. Firstly, the tracks are ordered by *popularity*, with the highest popularity first. The user defines the popularity of a track wh en initially entering the track into the system. If there is more than one track with the same popularity, those tracks are ordered by the record number, with the highest record number first. This means that the track from the most recent record would be played before the other tracks. This gives priority to the newer tracks in his collection. This means that although the actual popularity value of a track is never automatically decreased, the popularity of a track is indirectly decreased with the addit ion of newer tracks to the system.

If there is still no single best track i.e. there is more than one track with the same popularity and record number that has passed the constraints, the tracks are ordered by the *last played* date, with latest last played date first. The last played date of a track is updated automatically when the track is on a set list, and the user 'confirms' the set list will be played. The user 'confirms' a set list will be played by simply clicking on a button that executes an UPDATE SQL statement, updating the last played date.

This prioritisation was easy to achieve by adding the following ORDER BY clause at the end of the SQL query:

```
ORDER BY Track.Popularity DESC, Record.Record_Number DESC, Track.LastPlayed;
```

In the unlikely ci rcumstance that there is still no single best track after ordering by the *last played* date, the track is selected according to how the software orders the tracks.

Example

The following example will help to explain exactly how the constraints are applied: **Track X** is the first track inserted into the set list:

	Style	Speed	Mood	Popularity	Record No	Last Played
Track X	Big Beat	A	2	8	34	1/3/01

The user has selected a *Chilled* set type. This means that the mood must stay between 1 and 2. The most appropriate track must be selected from the following 10 tracks:

	Style	Speed	Mood	Popularity	Record No	Last Played
Track A	Nu Skool	A	1	8	23	1/3/01
Track B	Big Beat	**D**	2	9	34	23/3/01
Track C	**Techno**	A	1	6	12	2/12/00
Track D	Nu Skool	B	**4**	7	35	1/3/01
Track E	**Dark**	A	2	5	11	16/12/00
Track F	**Dark**	**D**	**5**	7	22	1/3/01
Track G	Nu Skool	**C**	2	6	32	2/12/00
Track H	Big Beat	A	2	8	45	1/3/01
Track I	Jump Up	A	1	7	21	1/3/01
Track J	Big Beat	B	1	8	17	23/3/01

The shaded tracks (tracks B - G) were rejected when the constr aints were applied. The values that violated the constraints are displayed in bold. Tracks C, E and F were rejected due to violating the style constraint. Tracks B, F and G all violated the speed constraint. These three tracks were all too fast to foll ow track X. Tracks A, H, I and J remained after the constraints had been enforced:

	Style	Speed	Mood	Popularity	Record No	Last Played
Track A	Nu Skool	A	1	8	23	1/3/01
Track H	**Big Beat**	**A**	**2**	**8**	**45**	**1/3/01**
Track I	Jump Up	A	1	7	21	1/3/01
Track J	Big Beat	B	1	8	45	23/3/01

These tracks were firstly ordered by popularity. Track I was rejected as the other three tracks all had higher popularity values. As all three tracks had popularity values of 8, they were ordered by the record number to find the most recent track. Tracks A and J had the highest record number, 45. This means the two tracks are from the same record. To select the better of these two tracks they were ordered by the last played date. **Track H** would be selected to follow **track X,** as this track had the latest last played date of the two.

5.1.2 Soft-Rule ‘Weighting’ Approach

This approach to generating a set list orders all tracks in the system by multiplying track attributes with carefully chosen parameters. I needed to write functions to give the attributes style, speed and mood numerical values. These functions compare how suitable a tracks *style*, *speed*, and *mood* is given the *style*, *speed*, and *mood* of the track that has just been selected.

The following code shows part of the *style* function (*styleOK*), where **x** is the style of the current track, and **y** is the style of the track being compared:

```
Function styleOK(x As String, y As String) As Double

If x = "Big Beat" And y = "Big Beat" Then
    styleOK = 0.8
ElseIf x = "Big Beat" And y = "Dark" Then
    styleOK = 0.3
ElseIf x = "Big Beat" And y = "Jump Up" Then
    styleOK = 0.7
ElseIf x = "Big Beat" And y = "Nu Skool" Then
    styleOK = 0.4
ElseIf x = "Big Beat" And y = "Techno" Then
    styleOK = 0.3
.
.
```
Fig 5.1

Please note: This section of the function only shows the values that are assigned if the current style is *Big Beat*. The complete function assigns values for all styles in the system.

The *speed* function (*speedOK*) for a Bangin’ set works in a similar way: assigning values between 0 and 1 given a tracks speed and a current track speed. For example, if the current speed is B, 0.1 is returned with A, 1 with B, 1 with C, 0.2 with D and 0.1 with E. A separate *mood* function had to be developed for each set type, as different values need to be assigned to different moods depending on the set type that has been selected. The complete style, speed and mood functions can be seen in Appendix G.

After defining the functions, the next step was to write a formula that would assign each track with a value, thus allowing the tracks to be ranked. Using the weighted sum approach I devised the following formula:

(moodParam * Mood) + (styleParam * Style) + (speedParam * Speed) + (popularityParam * Popularity) + (recordnumParam * Record_Number) + (lastplayedParam * Date()-LastPlayed)

The *moodParam* and *styleParam* etc. are the parameters that will be used. Obviously, each parameter will be assigned a value. For the mood, style and speed, the functions previously explained will be used.

Date() is a function that returns today's date. Subtracting the last played date from the current date gives the number of day's difference.

The following SQL query (Fig 5.2) uses the formula. As you can see the formula has been translated into SQL along with the functions, and put in the ORDER BY clause of the statement. The DESC at the end of the statement orders the tracks in descending order. After computation, the highest scoring track i.e. the most appropriate track, is found at the top.

```
SELECT *
FROM nextTrack
ORDER BY
('" & moodParam & "' * banginMood('" & arrayValue & "', Mood)) +
('" & speedParam & "' * speedOK('" & previousSpeed & "', Speed)) +
('" & styleParam & "' * styleOK('" & previousStyle & "', Style)) +
('" & popularityParam & "' * popularity) +
('" & recordnumParam & "' * Record_Number) +
('" & lastplayedParam & "' * (Date()-LastPlayed)) DESC;
```

Fig 5.2

This approach to generating a set list is *soft-ruled* as no track is ever completely ruled out of being played.

Example

Using the example from before I will show how the 'weightings' approach is applied. **Track X** is the first track inserted into the set list:

	Style	Speed	Mood	Popularity	Record No	Last Played
Track X	Big Beat	A	2	8	34	1/3/01

The user has selected a *Chilled* set type. The most appropriate track must be selected from the following 10 tracks:

	Style	Speed	Mood	Popularity	Record No	Last Played
Track A	Nu Skool	A	1	8	23	1/3/01
Track B	Big Beat	D	2	9	34	23/3/01
Track C	Techno	A	1	6	12	2/12/00
Track D	Nu Skool	B	4	7	35	1/3/01
Track E	Dark	A	2	5	11	16/12/00
Track F	Dark	D	5	7	22	1/3/01
Track G	Nu Skool	C	2	6	32	2/12/00
Track H	Big Beat	A	2	8	45	1/3/01
Track I	Jump Up	A	1	7	21	1/3/01
Track J	Big Beat	B	1	8	45	23/3/01

The following parameter values will be used for this example:

	moodParam	styleParam	speedParam	popularityParam	recordnumParam	lastplayedParam
Value	800	800	400	20	10	1

For this example the Date() function would return '5/4/01'.

Using the SQL query (Fig 5.2) and functions (Appendix I), a value can be computed for each of the 10 tracks, given track X. For track A the weighted sum would return the following value:

(800 * 1) + (800 * 1) + (400 * 0.4) + (20 * 8) + (10 * 23) + (1 * 35) = **2185**

The remaining 9 tracks were computed as a bove. The following table shows the values that were returned for each track:

	Style	**Speed**	**Mood**	**Popularity**	**Record No**	**Last Played**	**Total**
Track A	160	800	800	160	230	35	**2185**
Track B	320	80	800	180	340	13	**1733**
Track C	120	800	800	120	120	124	**2084**
Track D	160	800	0	140	350	35	**1485**
Track E	120	800	800	100	110	110	**2040**
Track F	120	80	-800	140	220	35	**-205**
Track G	160	160	800	120	320	124	**1684**
Track H	320	800	800	160	450	35	**2565**
Track I	280	800	800	140	210	35	**2265**
Track J	320	800	800	160	450	13	**2543**

The tracks are ranked as follows:

		Value
Rank 1	**Track H**	**2565**
Rank 2	Track J	2543
Rank 3	Track I	2265
Rank 4	Track A	2185
Rank 5	Track C	2084
Rank 6	Track E	2040
Rank 7	Track B	1733
Rank 8	Track G	1684
Rank 9	Track D	1485
Rank 10	Track F	-205

Track H would be selected to follow **track X** using the soft -ruled ‘weighting’ approach. The only four tracks that were left after the constraints were enforced using the hard-ruled approach were tracks A, H, I and J. These four tracks are all ranked in the t op 4 using the ‘weightings’ approach. Both approaches selected track H as the most appropriate track to follow track X.

5.2 Relative Merits of the Two Approaches

The 'weightings' approach allows the user to attach much more accurate values to preferences for the next track given a current track. If you look at Fig 5.1 you can see that, for example, if the current track has a *Big Beat* style, the most preferred style to follow would be *Big Beat* (reflected by the 0.8 value), followed by *Jump Up* (0.7), followed by *Nu Skool* (0.4), and so on. This accuracy is not possible using the hard-ruled approach. With that approach, if the current track has a *Big Beat* style, constraints are applied to say that the next track must either have a *Big Beat* style, *Jump Up* style, or *Nu Skool* style. Preferences for style are not stated. Tracks having a *Dark* or *Techno* style are completely rejected. Fig 5.1 shows that this is not the case using the weightings approach. Tracks with *Dark* or *Techno* styles are assigned 0.3, making them unlikely to appear, but still possible.

A major benefit of the 'weightings' approach is it can be fine-tuned to reflect how much weight each constraint should carry. This provides the user with much more flexibility, allowing him to generate different sets according to the weights used. The hard-rule approach cannot be tuned.

Another advantage of the 'weightings' approach is it always ensures the set list is filled according to the duration specified. This is not the case with the hard-rule approach. For example, track X is inserted into the set list. Track X has a *Dark* style and *D* speed. The hard-rule approach states that the next track must have a *Dark*, *Techno* or *Nu Skool* style, and a *D* or *E* speed. There is a track in the system that fits this (this track has mood 3). But, the user has selected a Bangin' set type, meaning that the mood of the next track must be *4*. This now means there is no track in the system that passes all three constraints. This would produce a message informing the user that the system has been unable to fill the set list with enough tracks.

With the 'weightings' approach, although a track may not strictly pass all constraints defined in the hard-rule approach, the most appropriate track is always computed. This is better under most circumstances, as it does not leave the user with incomplete set lists, although the hard-rule approach does always ensure that two incompatible tracks are never played together.

The weightings approach reflects much more closely how humans think. For example, in reality if track Y is perfect to follow track X, apart the fact that the two tracks styles do not match according to the style constraint, they would still be played together. The other track attributes 'out-weigh' the fact that the two styles may not be ideally matched. This is how the 'weightings' approach works. The hard-rule approach would simply reject track Y, and search for another track.

The overall project aim was to capture the knowledge of my end user using IT, and use this knowledge to automate the process of generating a set list. The 'weightings' approach is more successful at achieving this aim.

5.3 Role of the End User

After implementing and testing the two approaches the role of the end user was to asse ss which of the two approaches produced the best results. This involved getting the end user to generate a series of sets using each approach. He spent a good deal of time playing some of these sets, to see how the tracks were mixing together.

5.3.1 Fine-Tuning

To ensure that the end user was getting the best from the 'weightings' approach I spent a good deal of time with him fine-tuning the settings. This is explained in section 4.7.

I have implemented a weight settings control form, which gives the u ser flexibility, allowing him to alter the settings as he wishes (Fig 4.5). This form allows him to return to the default settings we decided upon, any time he wishes.

5.3.2 End User Feedback

After using both approaches my end user has decided that the ' weightings' approach is the better of the two approaches. Firstly, he preferred that the 'weightings' approach always gave a complete set list. The hard - rule approach sometimes leaves the set list incomplete, if none of the tracks in the system pass the constraints, requiring the user to add tracks to the set list. He was also impressed with the settings control form, as it gives him much more control over the type of set he generates each time.

Overall he identified that the 'weightings' approach was i ncluding a greater proportion of his better tracks in the set lists it generated.

Chapter 6 – Testing

6.1 Testing the System

Testing is essential to ensure every part of the system is functioning correctly. The system has been tested throughout its development i.e. as a new function was added to the system it was tested to ensure it was working correctly. Now the system is complete, the system needs to be tested as a whole.

6.1.1 Testing the Interface

The interface allows the user to enter data into the system, and search for specific data. The forms need to be thoroughly tested to ensure invalid data cannot be accepted by the system, and everything is working as required. The following checks were made:

1) The data accepted by a fie ld was valid. This involved attempting to enter invalid data into each field to ensure it could not be accepted. For example, trying to enter text into a date field. I found that no field would accept invalid data.

2) The fields won't accept null value s. I firstly attempted to save a null *Label* on the 'Add Label' sub-form. I found that the actually system tried to save the null value, causing an uninformative error message to be produced by Microsoft Access. After looking at the code I realised what I had done wrong. I currently set the default value for the field to "", which gave the appearance of a blank field. In the code I then checked to see if the field value was "". If so an error message was produced. I realised that this is fine if the u ser goes straight to the *Save Label* button. The problem comes when the user enters a label, deletes the label, and then attempts to save the label, because the "" would be deleted. To solve this problem I removed the default value, and used the `IsNull` function to check for a null value. This now works correctly. I needed to make this same update on all the other forms used for data input.

After updating the code, no field would accept a null value. When attempting to save a null value the system produces an informative error message informing the user they cannot leave any fields blank.

3) The primary key fields won't accept non -unique values. I firstly needed to ensure I couldn't save a non - unique *Record Number* to the 'Record' table. After attempti ng to enter a *Record Number* that already existed the system produced an error message.

I also needed to ensure that a non -unique neighbouring relationship wouldn't be saved to the system. After attempting to re-define a neighbouring relationship that a lready existed, the system didn't detect it, causing an uninformative error message being produced by Microsoft Access complaining about key violations. I needed to detect for this so an informative message could be produced to the user in future.

To achieve this I altered the code so it would search for the proposed neighbouring relationship, before attempting to save it to the system. If the neighbouring relationship already exists an error message is now produced. I also made this alteration on the *Generate Set List* form.

4) The command buttons all work as required. I clicked on every command button in the system and observed whether they were working as expected. I found that every command button in the system was working correctly.

5) The search facilities find the correct data. I needed to test each of the search combo-boxes to ensure they filtered the data correctly. Every search combo-box in the system filtered the data correctly.

6) The data displayed to the user cannot be edited. It is important that the user cannot edit the data that is being displayed to them, for example on the search screens. I attempted to edit every field, and found that some of the fields could actually be edited. To solve this I altered the properties of each field that could be edited.

The complete testing results can be seen in Appendix I.

6.1.2 Testing the Set List Generator

The set list generator needed to be fully tested to ensure every aspect of the generation process was working correctly. The following were tested:

The most appropriate starting tracks were getting selected

The first step in the set list generation process is to select the most appropriate **starting track**. The most appropriate starting track differs for the set type selected, so I tested each set type in turn. For testing I entered the following 4 tracks into the system, and generated a set list for each set type. The *popularity* value and *last start* date assigned for each of the four tracks were exactly the same. This means the track with the higher *record number* would be given preference over another track also having a suitable mood.

	Start Set	**Mood**	**Record No**
Track 1	Yes	1	56
Track 2	Yes	1	57
Track 3	Yes	2	58
Track 4	Yes	3	59

Bangin': This set requires the starting track to have mood 1. *Track 2* was selected. ✔

Chilled: This set requires the starting track to have mood 1 or 2. *Track 3* was selected. ✔

Club: This set requires the starting track to have mood between 2 and 4. *Track 4* was selected. ✔

Background: This set requires the starting track to have any mood. *Track 4* was selected. ✔

The start track selected for each set type was as expected.

The most appropriate finishing tracks were getting selected

To test the most appropriate finishing tracks were getting selected my plan was to generate a 30 -minute set, and analyse whether the most appropriate track had been selected to finish the set. To make this easier I set all tracks in the system to *Finish Set* = 'No', and then entered the following four tracks into the system. The *style*, *speed* and *popularity* values, and *finish* date assigned for each of the four tracks were exactly the same. I assigned the tracks low popularity values to ensure they were not selected before the end of the set. I again tested each set type in turn.

	Finish Set	**Mood**	**Record No**
Track 1	Yes	1	56
Track 2	Yes	2	57
Track 3	Yes	3	58
Track 4	Yes	4	59

Bangin': The finishing track would preferably have mood 4. *Track 4* was selected. ✔
Chilled: The finishing track would preferably have mood 1 or 2. *Track 2* was selected. ✔
Club: The finishing track would preferably have a mood between 2 and 4. *Track 4* was selected. ✔
Background: The finishing track could have any mood. *Track 4* was selected. ✔

The finishing track selected for each set type was as expected.

The starting style feature works

The *starting style* feature allows the user to select the style they want the starting track to be. I generated a series of set lists, changing the starting style each time. The starting track had the required style in each case.

The correct number of tracks is selected according to the duration selected

I estimated that a 30-minute set should include 8 tracks. I generated a 30-minute, and it included 8 tracks.

The weight settings affect the set generation in the correct way

Each constraint contributes different weights to the prioritisation of tracks. To ensure each constraint was functioning correctly I tested each one separately. I achieved this by lowering every other constraint to 0 on the *Custom Weight Settings* form, meaning they contributed nothing to the prioritisation, and increasing the constraint I was testing to 1000. I then generated a 40 -minute set and observed whether the tracks selected correctly followed the constraint I was enforcing.

Mood

The mood is selected according to the set type selected by the user, so I tested each set type individually:

Bangin': The mood of each track fit perfectly with the array for a 40-minute set. ✔
Chilled: The mood remained between 1 and 2 throughout. ✔
Club: The mood remained between 2 and 4 throughout. ✔
Background: Any mood was selected. ✔

Speed

After selecting a *Bangin'* set, the speed of the set gradually increased. ✔ For the other three set types the speed always either gradually increased or decreased. ✔

Style

I set the style of the starting track to be *Dark*. In selecting the next track to follow, the style function gives the highest values to those tracks also with a *Dark* style. Every track in the 40 -minute set I generated had a *Dark* style. ✔

Popularity

To make this easier I decreased the popularity value of all tracks in the system to 1. I then added 8 tracks, and assigned a higher popularity value to each new track I added. The tracks with the highest popularity were selected first. ✔

Last Played

I altered the *last played* dates of several tracks on the set list and generated a 30 -minute set. The tracks with the latest last played date were selected first. ✔

Record Number

I generated a 30-minute set, and the tracks from the highest records were selected first. ✔

The most appropriate track was selected each time

I needed to ensure the track being selected each time was actually the most appropriate. To test this my plan was to generate a 30 -minute set, and then analyse the entire set list to ensu re the most appropriate track was selected each time. To make this easier I lowered the *popularity* values of all the tracks in the system. I then entered 8 new tracks into the system, and gave the tracks high *popularity* values. From the *style*, *speed* and *mood* etc. I assigned to each track, I was able to predict the order in which the tracks would appear on the set list. I generated a 30-minute set using the *Bangin'* set type. The tracks appeared in the order I had predicted. The report for the generated set list can be seen in Appendix I.

The random number generation was working

To give variety to the set lists the code generates a random number between 1 and 20, and executes a certain section of code depending on the number that is generated:

A) *Between 1 and 10*: Insert the most appropriate track from the current playlist.
B) *Between 11 and 19* : Insert a neighbouring track of the previous track, if the previous track has a neighbouring track.
C) *20*: Insert the most appropriate classic track.

To test this was working correctly I updated the code so that a message box would appear each time and inform me which of the three sections of code had been accessed. I generated 5 60-minute sets, and recorded the results:

	1	**2**	**3**	**4**	**5**	**Total**	**%**
A	11	13	13	12	12	61	**54**
B	14	7	8	8	11	48	**42**
C	0	0	2	2	0	4	**4**
Totals	25	20	23	22	23	113	100

In theory, A has a 50% probability of occurring, B 45% probability and C 5% probability. The results I obtained are very close to these expected results.

The neighbouring tracks feature was working

To test whether the neighbouring tracks feature was working correctly I generated a 30-minute set, and analysed each of the 8 tracks to see if they had a neighbouring track. If so, I then checked to see whether the neighbouring track had been either included on the set list, or included on the currently un-selected neighbouring track list. Of the 8 tracks on the set list I found 2 had neighbouring tracks:

Track 2 had 2 neighbouring tracks. 1 was inserted as a neighbouring track i.e. as track 3, and the other appeared on the currently un-selected neighbouring track list. ✔

Track 5 had 1 neighbouring track. This track was inserted as a neighbouring track i.e. as track 6.✔

The classic track feature was working

As the probability of a classic track appearing is low, I altered the random number generation to highly increase the probability of a classic track appearing. This allowed me to better test the following:

a) A classic track is never included in the first 30 minutes of a set.

In the series of 60-minute sets I generated a classic track never appeared within the first 30 minutes.✔

b) Classic tracks are never played straight after each other.

There was never a case where two classic tracks were played straight after each other. ✔

c) The include classic check box was working.

The *Include Classic Tracks* checkbox allows the user to define whether they want classic tracks including in the set list. To test this I unchecked the box, and generated a series of 60-minute sets. No classic tracks were ever included. ✔

The re-use feature was working

The theory behind the *Re-Use* attribute attached to each record is, if *Re-Use* is set to 'No', when a track from that record has been selected for the set list, no other track from that rec ord can be selected. If *Re-Use* is set to 'Yes', when a track from the record has been selected for the set list, other tracks from that record can also be selected. This effect is achieved by running an SQL UPDATE query when a track has been inserted int o the set list, to set the *Selected* value to 'Yes' on all other tracks from the record if *Re-Use* is set to 'No' on their record.

To test the re -use feature was working I generated a 60 -minute set. I observed it wasn't working correctly, as sometimes more than one track from a record having *Re-Use* set to 'No' was been included on the same set list. After debugging I realised the SQL UPDATE query was wrong:

```
UPDATE Track SET Selected = yes WHERE Record_Number = RecordNumberBox.value AND
Reuse = no;
```

The problem was, the query didn't join the 'Track' table with the 'Record' table, and therefore had no way of knowing whether the track was from a record that couldn't be reused.

The following is the updated query:

```
UPDATE Track INNER JOIN record ON Track.Record_Number  = Record.Record_Number
SET Selected = yes WHERE Record.Record_Number = RecordNumberBox.value AND
Record.Reuse = no;
```

I repeated the testing after updating the query, and found it was working correctly.

The warnings appear at the correct time

The system should provide the following warnings:

No Starting Track

This warning appears to inform the user there was no suitable starting track. To test this was working I ensured there was no suitable starting track, and then generated a set. The system firstly provided a message box, and then the warning appeared on-screen. ✔

Duration less than specified

This warning appears to inform the user the specified duration could not be filled. To test this I generated a 500-minute set. The system firstly provided a message box, and then the warning appeared on-screen. ✔

No Finishing Track

This warning appears to inform the user there was no suitable finishing track. To test this was working I ensured there was no suitable finishing track, and then generated a set. The system firstly provided a message box, and then the warning appeared on-screen. ✔

Record(s) Re-used

This warning appears to inform the user more than one track from the same record is included on the set list. To test this warning was working correctly I generated a 60-minute set, and went through the set list to see whether the same record appeared more than once. A record appeared twice, and the warning appeared on-screen. ✔ I then generated a 30-minute set, and noticed that no record had been re-used. On this occasion the warning was not produced. ✔

The last played date was updated correctly

When a user selects to a*ccept* a set list, the *Last* P*layed* date of each track on the set list should be updated to the G*ig Date* entered. The *Last Start* date of the starting track, and the *Last Finish* date of the finishing track should also be updated. To test this I generated a 30-minute set, clicked on the *Accept Set List* button, and checked the dates of the tracks on the set list. All dates had been updated as required. ✔

Results of System Testing

I can conclude that every aspect of the system has been thoroughly tested. The user cannot enter invalid data, and the set list generator is now working as required. All errors detected during testing have been rectified.

6.2 User Acceptance Testing

User acceptance testing is an essential activity that should be undertaken to ensure the user is getting what they actually wanted from the system. This involved getting the end user to test the sys tem against the system requirements we set. The results of this testing can be seen in Appendix J. They show the system I have developed provides the functionality required by the user.

Chapter 7 – Evaluation and Future Developments

7.1 System Evaluation

Overall, I believe I have developed a system that is fundamentally correct and user-friendly. The overall aim for the project was "*to capture the knowledge of a DJ using IT, and use this knowledge to automate the process of generating a set list*". I firmly believe this aim has been achieved. The first challenge was to devise a knowledge capture model. I came up with two such models, a hard-rule model and a weightings model. The next challenge was to use these models to actually capture the knowledge of the DJ. Both models successfully achieved this. The weightings model became the preferred model after fine-tuning the weights with the DJ, producing set lists very similar to set lists the DJ would prepare.

The user acceptance testing confirmed that all functional and non-functional requirements have been met. The end user has tested the user interface. He found it very useable, and was able to successfully enter a number of records into the system. The DSDM methodology ensured the user was actively involved in the development of the system. This gave me the flexibility to deal with the changing system requirements throughout the course of the project, ensuring the end product was what the end user actually wanted.

A limiting factor for most systems is data input. In the system I have developed, each track must have accurate values for its attributes, such as mood and style, in order to accurately capture that the end users knowledge. It could be time consuming for the end user to ensure that all tracks in the system are defined accurately. Another restriction with the system is the user has to manually remove the tracks he doesn't want appearing on the set lists, from the current playlist. This could again be time consuming. A major reason for developing the system was to save the end user time; yet having to spend time keeping a system up-to date creates work in itself!

Only about a third of my end users records are currently stored in the system. The system is currently generating the kind of set lists he is looking for, but after entering all records from the collection there is no guarantee that the system will continue to be as successful. He is optimistic that the system will be of real use to him, but only time will tell. He will definitely continue to use the input and retrieval section of the system, as it allows him to keep an accurate track of all the records in his collection.

He did raise an important point: the system could never fully determine the tracks that are played during a set, as many other factors come into it in reality. For example, the crowd's reaction to certain tracks may mean that tracks other than the ones planned on the set list should be played. Playing a DJ set could never be fully automated!

7.2 Future Enhancements

There are a number of features I would like to add to my current system, if time permitted.

Firstly, I would add a graph feature to the system, whereby the user would be presented with a blank graph of mood against time, and would be able to draw a custom graph for a set. This would determine the mood of the tracks selected, and give the user maximum flexibility to generate the exact set they are looking to play.

It would also be nice to provide full on-line help, which would be specific to the area of the system the user is currently having problems with. The user manual I developed is useful, but users generally prefer on-line help as opposed to reading a manual.

In my mid-project report I stated the following objective should be completed if time permitted:

Design and implement a system that will extract specific records from a long list of newly released records, using keywords.

The lists of newly released records are sent to my end user via e-mail each month as text files, and give information such as the label and artist of the new records to be released. Unfortunately, I didn't have time to provide a solution to this problem, but given more time it would provide me with an excellent opportunity to link the system I have developed with the key word searching of the text files. The search criteria could be taken from the database, and used in the search. For example, software could be written to find if any of the label's stored in the database are found in the text file.

In response to the data input problem, a possible enhancement would be to use auto-analysis software, which could for example, be trained to identify a tracks speed and style. Such software could be used when inputting a new record into the system. This would somewhat reduce the data input required by the user of the system.

References

Avison, D and Fitzgerald, G, *Information Systems Development: Methodologies, Techniques and Tools*, Second Edition, McGraw-Hill, 1995

Elmasri, R and Navathe, S, *Fundamentals of Database Systems: Third Edition*, Addison-Wesley, 1999

Gaskell, R, *A WWW Real Ale Guide to Leeds,* 1998

Johnson, P, *Human Computer Interaction: psychology, task analysis and software engineering*, McGraw-Hill Book Company, 1992

Lau, L and McCormack, J, *IN22 (Fourth Generation Languages and Prototyping) Notes*, School of Computing, University of Leeds, 2000

Linscott, G, *A User-Friendly Information System and Web Site for Armour Services Ltd*, 1999

Mott, P and Roberts, S, *DB11 (Introduction to Databases) Notes*, School of Computing, University of Leeds, 1998

Roberts, S, *DB21 (Database Principles and Practice) Notes*, School of Computing, University of Leeds, 1999

Sutcliffe, S, *An Online Mortgage Quotation and Application System*, 1999

On-Line Resources

Microsoft Corporation,	http://www.microsoft.com (last visit: 6/12/2000)
Song Librarian 2.0,	http://www.hitsquad.com/smm/programs/The_Song_Librarian/ (last visit: 9/11/2000)

Appendix A – Project Reflection

The project proved to be challenging at times, but overall it proved to be both an enjoyable and interesting experience. I have fulfilled all objectives of this project I initially set out to achieve, within the deadline I was given. I am personally very pleased with the final system I have developed, and am hopeful it will be put to some use.

I have gained a number of valuable skills through the course of the project, both technical and non-technical, that I will be able to put to use in my future career. Technical skills include database design, Microsoft Access, SQL, Visual Basic, and technical report writing. Non-technical skills cover things such as time management, and conducting meetings.

I was pleased with my selection of tools to develop the system. Microsoft Access proved to be a very useful tool for prototyping. I found I could quickly develop an interface using the controls provided by the software, such as command buttons and combo boxes, and later add functionality using VBA.

As my user was also very busy it made it quite difficult to find time for meetings, as he obviously had more important priorities than my system. This meant that when we did get chance to meet it was essential that the meeting was planned fully beforehand, to ensure I obtained everything from the meeting I required.

The main criticism I have of the way I developed the system was my failure to comment the code during implementation. This meant that when revisiting the code months later I sometimes found it difficult to understand what the code was actually doing, or why I had chosen to do it a particular way. I eventually managed to comment the code, but I could have saved myself a lot of effort by commenting as I was coding.

Appendix B – Project Schedule

Appendix C – Data Dictionary

Label

Field Name	Data Type	Description
Label_ID	AutoNumber	Primary key, unique label identifier
Label	Text	Name of the label

Record

Field Name	Data Type	Description
Record_Number	Number	Primary key, unique number given to each record
Record_Title	Text	Title of the record
Label_ID	Number	Foreign key from the 'Label' table
Reuse	Yes/No	Can more then one track from the record be selected on the same set list?

Track

Field Name	Data Type	Description
Track_ID	AutoNumber	Primary key, unique track identifier
Record_Number	Number	Foreign key from the 'Record' table
Track_Title	Text	Title of the track
Mix	Text	Mix of the track
Side	Text	The side of the record the track is on
Style	Text	The style of the track
Speed	Text	An estimated speed of the track in BPM
Popularity	Number	A number between 1 and 9 reflecting how popular the track is
Mood	Number	A number between 1 and 5 reflecting the stage in the set where the track is best played (1 = start of set, 5 = end of set)
Start_Set	Yes/No	Would the track ever start a set?
Finish_Set	Yes/No	Would the track ever finish a set?
CurrentPlaylist	Yes/No	Is the track on the current playlist? Tracks on the current playlist can be selected for each set list generated
Classic	Yes/No	Is the track a classic? A classic track can be selected once in a while
LastPlayed	Date/Time	When was the track last played?
LastStart	Date/Time	When did the track last start a set?
LastFinish	Date/Time	When did the track last finish a set?
Selected	Yes/No	Is the track currently on the set list?

Artist

Field Name	Data Type	Description
Artist_ID	AutoNumber	Primary key, unique artist identifier
Artist	Text	Name of the artist

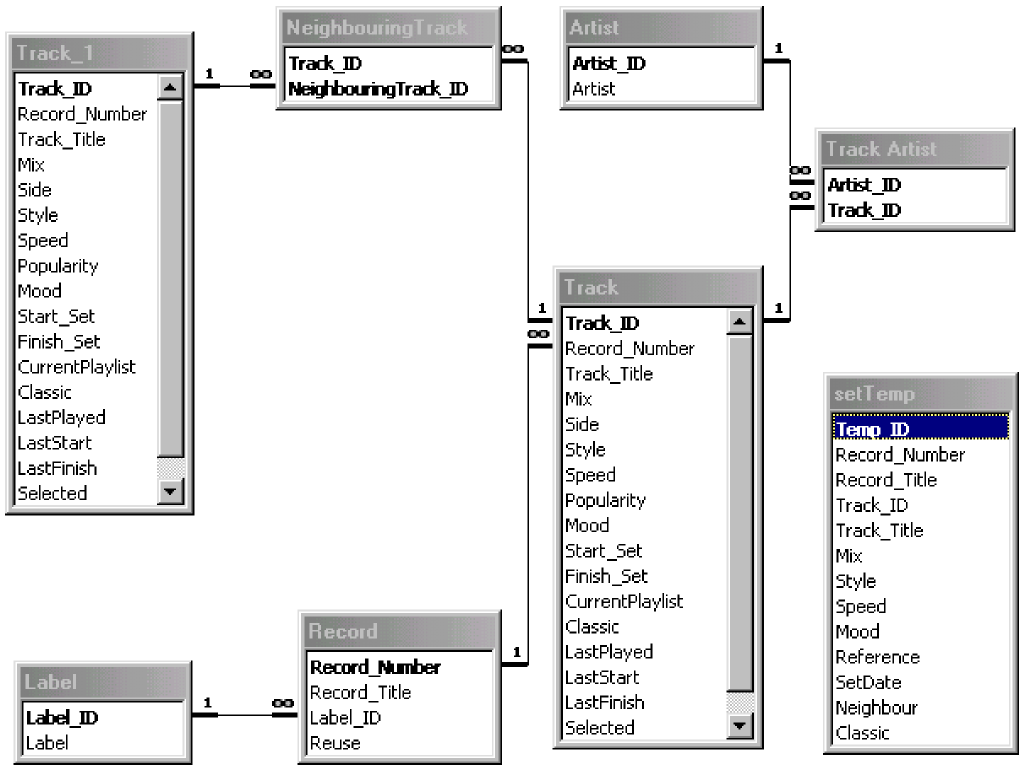
Track Artist

Field Name	Data Type	Description
Artist_ID	Number	Primary key from the 'Artist' table
Track_ID	Number	Primary key from the 'Track' table

NeighbouringTrack

Field Name	Data Type	Description
Track_ID	Number	Primary key, first track in the neighbouring relationship
NeighbouringTrack_ID	Number	Primary key, second track in the neighbouring relationship

Appendix D – Database Relationship Diagram



The ‘Track’ table appears twice on the diagram. This is to ensure referential integrity is upheld between both tracks in the ‘NeighbouringTrack’ table.

Appendix E – System Interface

Appendix F – Hard-Rule Constraints

Style Constraint

Style	Follow-on Style
Dark	Dark
Dark	Nu Skool
Dark	Techno
Big Beat	Big Beat
Big Beat	Jump Up
Big Beat	Nu Skool
Jump Up	Big Beat
Jump Up	Jump Up
Jump Up	Nu Skool
Nu Skool	Dark
Nu Skool	Big Beat
Nu Skool	Jump Up
Nu Skool	Nu Skool
Nu Skool	Techno
Techno	Dark
Techno	Nu Skool
Techno	Techno

Bangin’ Speed Constraint

Speed	Follow-on Speed
A	A
A	B
B	B
B	C
C	C
C	D
D	D
D	E
E	E

Normal Speed Constraint

Speed	Follow-on Speed
A	A
A	B
B	A
B	B
B	C
C	B
C	C
C	D
D	C
D	D
D	E
E	D
E	E

Appendix G – Weightings Functions

Style Function

```
If x = "Big Beat" And y = "Big Beat" Then
    styleOK = 0.8

ElseIf x = "Big Beat" And y = "Dark" Then
    styleOK = 0.3

ElseIf x = "Big Beat" And y = "Jump Up" Then
    styleOK = 0.7

ElseIf x = "Big Beat" And y = "Nu Skool" Then
    styleOK = 0.4

ElseIf x = "Big Beat" And y = "Techno" Then
    styleOK = 0.3


ElseIf x = "Dark" And y = "Big Beat" Then
    styleOK = 0.4

ElseIf x = "Dark" And y = "Dark" Then
    styleOK = 0.9

ElseIf x = "Dark" And y = "Jump Up" Then
    styleOK = 0.7

ElseIf x = "Dark" And y = "Nu Skool" Then
    styleOK = 0.7

ElseIf x = "Dark" And y = "Techno" Then
    styleOK = 0.5


ElseIf x = "Jump Up" And y = "Big Beat" Then
    styleOK = 0.5

ElseIf x = "Jump Up" And y = "Dark" Then
    styleOK = 0.5

ElseIf x = "Jump Up" And y = "Jump Up" Then
    styleOK = 0.8

ElseIf x = "Jump Up" And y = "Nu Skool" Then
    styleOK = 0.6

ElseIf x = "Jump Up" And y = "Techno" Then
    styleOK = 0.3


ElseIf x = "Nu Skool" And y = "Big Beat" Then
    styleOK = 0.6

ElseIf x = "Nu Skool" And y = "Dark" Then
    styleOK = 0.5

ElseIf x = "Nu Skool" And y = "Jump Up" Then
    styleOK = 0.8

ElseIf x = "Nu Skool" And y = "Nu Skool" Then
    styleOK = 0.9

ElseIf x = "Nu Skool" And y = "Techno" Then
    styleOK = 0.3

ElseIf x = "Techno" And y = "Big Beat" Then
    styleOK = 0.8

ElseIf x = "Techno" And y = "Dark" Then
```

```
    styleOK = 0.7

ElseIf x = "Techno" And y = "Jump Up" Then
    styleOK = 0.3

ElseIf x = "Techno" And y = "Nu Skool" Then
    styleOK = 0.6

ElseIf x = "Techno" And y = "Techno" Then
    styleOK = 0.8

End If
End Function
```

Bangin' Speed Function

```
Function speedOK(x As String, y As String) As Double

If x = "A" And y = "A" Then
    speedOK = 1

ElseIf x = "A" And y = "B" Then
    speedOK = 1

ElseIf x = "A" And y = "C" Then
    speedOK = 0.2

ElseIf x = "A" And y = "D" Then
    speedOK = 0.1

ElseIf x = "A" And y = "E" Then
    speedOK = 0


ElseIf x = "B" And y = "A" Then
    speedOK = 0.1

ElseIf x = "B" And y = "B" Then
    speedOK = 1

ElseIf x = "B" And y = "C" Then
    speedOK = 1

ElseIf x = "B" And y = "D" Then
    speedOK = 0.2

ElseIf x = "B" And y = "E" Then
    speedOK = 0.1


ElseIf x = "C" And y = "A" Then
    speedOK = 0

ElseIf x = "C" And y = "B" Then
    speedOK = 0.1

ElseIf x = "C" And y = "C" Then
    speedOK = 1

ElseIf x = "C" And y = "D" Then
    speedOK = 1

ElseIf x = "C" And y = "E" Then
    speedOK = 0.2


ElseIf x = "D" And y = "A" Then
    speedOK = 0
```

```
ElseIf x = "D" And y = "B" Then
    speedOK = 0

ElseIf x = "D" And y = "C" Then
    speedOK = 0.1

ElseIf x = "D" And y = "D" Then
    speedOK = 1

ElseIf x = "D" And y = "E" Then
    speedOK = 1


ElseIf x = "E" And y = "A" Then
    speedOK = 0

ElseIf x = "E" And y = "B" Then
    speedOK = 0

ElseIf x = "E" And y = "C" Then
    speedOK = 0.1

ElseIf x = "E" And y = "D" Then
    speedOK = 0.2

ElseIf x = "E" And y = "E" Then
    speedOK = 1

End If
End Function
```

Normal Speed Function

```
Function normalSpeedOK(x As String, y As String) As Double

If x = "A" And y = "A" Then
    normalSpeedOK = 1

ElseIf x = "A" And y = "B" Then
    normalSpeedOK = 1

ElseIf x = "A" And y = "C" Then
    normalSpeedOK = 0.2

ElseIf x = "A" And y = "D" Then
    normalSpeedOK = 0.1

ElseIf x = "A" And y = "E" Then
    normalSpeedOK = 0


ElseIf x = "B" And y = "A" Then
    normalSpeedOK = 1

ElseIf x = "B" And y = "B" Then
    normalSpeedOK = 1

ElseIf x = "B" And y = "C" Then
    normalSpeedOK = 1

ElseIf x = "B" And y = "D" Then
    normalSpeedOK = 0.2

ElseIf x = "B" And y = "E" Then
    normalSpeedOK = 0.1


ElseIf x = "C" And y = "A" Then
    normalSpeedOK = 0.1
```

```
ElseIf x = "C" And y = "B" Then
    normalSpeedOK = 1

ElseIf x = "C" And y = "C" Then
    normalSpeedOK = 1

ElseIf x = "C" And y = "D" Then
    normalSpeedOK = 1

ElseIf x = "C" And y = "E" Then
    normalSpeedOK = 0.2


ElseIf x = "D" And y = "A" Then
    normalSpeedOK = 0

ElseIf x = "D" And y = "B" Then
    normalSpeedOK = 0.1

ElseIf x = "D" And y = "C" Then
    normalSpeedOK = 1

ElseIf x = "D" And y = "D" Then
    normalSpeedOK = 1

ElseIf x = "D" And y = "E" Then
    normalSpeedOK = 1


ElseIf x = "E" And y = "A" Then
    normalSpeedOK = 0

ElseIf x = "E" And y = "B" Then
    normalSpeedOK = 0.1

ElseIf x = "E" And y = "C" Then
    normalSpeedOK = 0.2

ElseIf x = "E" And y = "D" Then
    normalSpeedOK = 1

ElseIf x = "E" And y = "E" Then
    normalSpeedOK = 1

End If
End Function
```

Bangin’ Mood Function

```
Function banginMood(x, y) As Double

If x = "1" And y = "1" Then
    banginMood = 1

ElseIf x = "1" And y = "2" Then
    banginMood = 0.2

ElseIf x = "1" And y = "3" Then
    banginMood = 0

ElseIf x = "1" And y = "4" Then
    banginMood = 0

ElseIf x = "1" And y = "5" Then
    banginMood = 0


ElseIf x = "2" And y = "1" Then
    banginMood = 0
```

```
ElseIf x = "2" And y = "2" Then
    banginMood = 1

ElseIf x = "2" And y = "3" Then
    banginMood = 0.2

ElseIf x = "2" And y = "4" Then
    banginMood = 0

ElseIf x = "2" And y = "5" Then
    banginMood = 0


ElseIf x = "3" And y = "1" Then
    banginMood = 0

ElseIf x = "3" And y = "2" Then
    banginMood = 0

ElseIf x = "3" And y = "3" Then
    banginMood = 1

ElseIf x = "3" And y = "4" Then
    banginMood = 0.2

ElseIf x = "3" And y = "5" Then
    banginMood = 0


ElseIf x = "4" And y = "1" Then
    banginMood = 0

ElseIf x = "4" And y = "2" Then
    banginMood = 0

ElseIf x = "4" And y = "3" Then
    banginMood = 0

ElseIf x = "4" And y = "4" Then
    banginMood = 1

ElseIf x = "4" And y = "5" Then
    banginMood = 0.2


ElseIf x = "5" And y = "1" Then
    banginMood = 0

ElseIf x = "5" And y = "2" Then
    banginMood = 0

ElseIf x = "5" And y = "3" Then
    banginMood = 0

ElseIf x = "5" And y = "4" Then
    banginMood = 0.1

ElseIf x = "5" And y = "5" Then
    banginMood = 1

End If
End Function
```

Chilled Mood Function

```
Function chilledMood(x) As Double

If x = "1" Then
    chilledMood = 1
```

```
ElseIf x = "2" Then
    chilledMood = 1

ElseIf x = "3" Then
    chilledMood = 0.5

ElseIf x = "4" Then
    chilledMood = 0

ElseIf x = "5" Then
    chilledMood = -1

End If
End Function
```

Club Mood Function

```
Function clubMood(x As Integer) As Double

If x = "1" Then
    clubMood = 0

ElseIf x = "2" Then
    clubMood = 1

ElseIf x = "3" Then
    clubMood = 1

ElseIf x = "4" Then
    clubMood = 1

ElseIf x = "5" Then
    clubMood = 0

End If
End Function
```

Appendix H – Set List Report

Appendix I – System Testing

Main Menu

Action	Expected Result	Actual Result
Click on the *Input New Record* button.	The *Add Record* form will be opened.	As expected.
Click on the *Add Track to a Record* button.	The *Find Record Extra* form will be opened.	As expected.
Click on the *Find Record* button.	The *Find Record* form will be opened.	As expected.
Click on the *Find / Update Track* button.	The *Find Track* form will be opened.	As expected.
Click on the *Generate Set List* button.	The *Generate Set List* form will be opened. The *Custom Weight Settings* form will be opened, but made invisible.	As expected. As expected.
Click on the *View Neighbouring Tracks* button.	The *Display Neighbouring Tracks* form will be opened.	As expected.
Click on the *Exit Application* button.	The application will be closed.	As expected.

Add Record

Action	Expected Result	Actual Result
On form-open.	The value in the *Record Number* field will be the value of the maximum record number in the system + 1.	As expected.
Click on the *Add New Label* button.	The *Add Label* sub-form will be made visible, and the caption on the button will change to *Cancel*. All controls not on the sub-form will be disabled.	As expected. As expected.
Click on the *Cancel* button.	The *Add Label* sub-form will be made invisible, and the caption on the button will change back to *Add New Label.* All controls not on the sub-form will be re-enabled.	As expected. As expected.
Enter a *Label* on the *Add Label* sub-form, and click on the *Cancel* button.	The label will not be saved to the system.	As expected.
Click on the *Save Label* button without entering a label.	An error message will be produced.	As expected.
Enter a label, delete the label, and then click on the *Save Label* button.	An error message will be produced.	The system didn't detect the field had been left null, and attempted to save a null label to the system, causing an error. See comments for a detailed explanation and solution.

Enter a *Label* on the *Add Label* sub-form, and click on the *Save Label* button.	The label will be saved to the system. This label will now appear in the *Label* combo-box on the *Add Record* form.	As expected.
Enter a *Label* that isn't a value in the combo-box.	An error message will be produced.	As expected.
Select a *Label* from the combo-box.	The value will be accepted.	As expected.
Enter text into the *Record Number* field.	An error message will be produced.	As expected.
Enter a unique *Record Number*, leave the *Title* field blank and click on the *Save Record* button.	An error message will be produced.	As expected.
Enter a *Record Number* that already exists and click on the *Save Record* button.	An error message will be produced.	As expected.
Enter a valid *Label*, *Record Number* and *Title* and click on the *Save Record* button	The record will be saved to the system, and the *Add Track* form will be opened displaying the *Title* and *Record Number* of the record. The *Add Record* form will be made invisible.	As expected. As expected.
Click on the *Cancel* button.	The *Add Record* form will be closed, and no record will be saved to the system. You will be returned to the *Main Menu*.	As expected.

Comments
To ensure a null value could not be accepted in the *Add Label* sub-form I set the default value to "", which gave the appearance of a blank field. In the code I then checked to see if the field value was "". If so an error message was produced. After testing I have realised that this is fine if the user goes straight to the *Save Label* button. The problem comes when the user enters a label, deletes the label, and then attempts to save the label, because the "" would be deleted. To solve this problem I remove d the default value, and used the `IsNull` function to check for a null value. This now works correctly. I have made the same update on all the other forms used for data input.

Add Track

Action	Expected Result	Actual Result
On form-open.	The title of record will be set as the *Title* default, and the *Mix* default will be "ORIGINAL".	As expected.
Enter a value that doesn't exist in a combo-box.	An error message will be produced.	As expected.
Select valid values from each combo-box.	They will be accepted.	As expected.
Select a valid *Speed*.	The speed in BPM will be displayed next to the combo-box.	As expected.
Attempt to leave each field blank, and click on the save button.	An error message will be produced.	As expected.
Enter valid values into each field and click on the *Save* button.	The track will be saved to the system, and the *Add Artist* form will be opened displaying the *Title* of the track. The *Add Track* form will be made invisible.	As expected. As expected.

Click on the *Cancel* button.	The *Add Track* form and *Add Record* form will be closed, and the track will not be saved to the system.	As expected.

Add Artist

Action	Expected Result	Actual Result
Click on the *Add New Artist* button.	The *Add Artist* sub-form will be made visible, and the caption on the button will change to *Cancel*. All controls not on the sub-form will be disabled.	As expected.
Click on the *Cancel* button.	The *Add Artist* sub-form will be made invisible, and the caption on the button will change back to *Add New Artist.* All controls not on the sub-form will be re-enabled.	As expected. As expected.
Enter an *Artist* on the *Add Label* sub-form, and click on the *Cancel* button.	The artist will not be saved to the system.	As expected.
Click on the *Save Artist* button without entering an artist.	An error message will be produced.	As expected.
Enter an *Artist* on the *Add Artist* sub-form, and click on the *Save Artist* button.	The artist will be saved to the system. This artist will now appear in the *Artist* combo-box on the *Add Artist* form.	As expected.
Enter an *Artist* that isn't a value in the combo-box.	An error message will be produced.	As expected.
Select an *Artist* from the combo-box and click on the *Add Additional Artist* button.	The artist will be saved to the track, and artist combo-box will be cleared and given the focus.	As expected.
Select an *Artist* from the combo-box and click on the *Save Artist to Track* button.	The artist will be saved to the track, and the *Add Artist*, and *Add Track* forms will be closed. The *Add Record* form will remain, with all fields disabled. The *Add New Record* button will be visible.	As expected. As expected. As expected.
Click on the *Cancel* button.	No artist will be saved to the track. The track will be deleted from the system, as every track must have an artist. The *Add Artist* and *Add Track* forms will be closed. The *Add Record* form will remain, with all fields disabled. The *Add New Record* button will appear.	As expected. As expected. As expected. As expected.

Add Record (after a track has been saved to a record)

Action	Expected Result	Actual Result
On return to form.	The *Add New Label* button is invisible. The *Label*, *Record Number*, Title, and *Re-use* fields are not enabled. The *Add New Record* button is visible.	As expected. The *Re-use* field was still enabled, so I corrected the code. As expected.
Click on the *Add Track* button.	The *Add Track* form will be opened. The *Add Record* form will be made invisible.	As expected. As expected.
Click on the *Cancel* button.	The *Add Record* form will be closed, and you will be returned to the *Main Menu*.	As expected.

Find Record Extra

Action	Expected Result	Actual Result
Select an option from any combo-box.	The *Display All Records* button will become visible.	As expected.
Select an option from the *Title* combo-box.	The records will be filtered according to the title selected.	As expected.
Select an option from the *Label* combo-box.	The records will be filtered according to the label selected.	As expected.
Select an option from the *Re-use* combo-box.	The records will be filtered according to the selection.	As expected.
Click on the *Display All Records* button.	All records will be displayed, and the combo-boxes will be cleared.	As expected.
Attempt to update the *Record Number*.	An update will not be allowed.	As expected.
Attempt to update the *Record Title*.	An update will not be allowed.	As expected.
Attempt to update the *Label*.	An update will not be allowed.	As expected.
Click on the *Re-use* field.	If it is currently unselected it will become selected, and if it is currently selected it will become unselected.	As expected.
Click on the *Delete* button.	The system will ask for confirmation. If the selection is 'no' the confirmation box will simply close. If the selection is 'yes' that record, and any associated tracks will be deleted from the system.	As expected. As expected. As expected. After deletion the *Find Record* form was opened. After looking at the code I realised I had made reference to the wrong form, so I corrected it.

Double-Click on the *Record Title*.	The *Add Track Extra* form will be opened displaying the *Title* and *Record Number* of the record. The *Find Record Extra* form will be made invisible.	As expected. As expected.
Click on the *Exit* button.	The *Find Record Extra* form will be closed, and you will be returned to the *Main Menu*.	As expected.

Add Track Extra

Action	Expected Result	Actual Result
On form-open.	The title of record will be set as the *Title* default, and the *Mix* default will be "ORIGINAL".	As expected.
Enter a value that doesn't exist in a combo-box.	An error message will be produced.	As expected.
Select valid values from each combo-box.	They will be accepted.	As expected.
Select a valid *Speed*.	The speed in BPM will be displayed next to the combo-box.	As expected.
Attempt to leave each field blank, and click on the *Save* button.	An error message will be produced.	As expected.
Enter valid values into each field and click on the *Save* button.	The track will be saved to the system, and the *Add Artist Extra* form will be opened displaying the *Title* of the track. The *Add Track Extra* form will be made invisible.	As expected. As expected.
Click on the *Cancel* button.	The *Add Track Extra* form will be closed, and the track will not be saved to the system. You will be returned to the *Find Record Extra* form.	As expected.

Add Artist Extra

Action	Expected Result	Actual Result
Click on the *Add New Artist* button.	The *Add Artist* sub-form will be made visible, and the caption on the button will change to *Cancel*. All controls not on the sub-form will be disabled.	As expected.
Click on the *Cancel* button.	The *Add Artist* sub-form will be made invisible, and the caption on the button will change back to *Add New Artist.* All controls not on the sub-form will be re-enabled.	As expected. As expected.
Enter an *Artist* on the *Add Label* sub-form, and click on the *Cancel* button.	The artist will not be saved to the system.	As expected.
Click on the *Save Artist* button without entering an artist.	An error message will be produced.	As expected.

Enter an *Artist* on the *Add Artist* sub-form, and click on the *Save Artist* button.	The artist will be saved to the system. This artist will now appear in the *Artist* combo-box on the *Add Artist Extra* form.	As expected.
Enter an *Artist* that isn't a value in the combo-box.	An error message will be produced.	As expected.
Select an *Artist* from the combo-box and click on the *Add Additional Artist* button.	The artist will be saved to the track, and artist combo-box will be cleared and given the focus.	As expected.
Select an *Artist* from the combo-box and click on the *Save Artist to Track* button.	The artist will be saved to the track, and the *Add Artist Extra*, and *Add Track Extra* forms will be closed. The *Find Record Extra* form will remain.	As expected. As expected.
Click on the *Cancel* button.	No artist will be saved to the track. The track will be deleted from the system, as every track must have an artist. The *Add Artist Extra* and *Add Track Extra* forms will be closed. The *Find Record Extra* form will remain.	As expected. As expected. As expected. As expected.

Find Record

Action	Expected Result	Actual Result
Select an option from any combo-box.	The *Display All Records* button will become visible.	As expected.
Select an option from the *Title* combo-box.	The records will be filtered according to the title selected.	As expected.
Select an option from the *Label* combo-box.	The records will be filtered according to the label selected.	As expected.
Select an option from the *Re-use* combo-box.	The records will be filtered according to the selection.	As expected.
Click on the *Display All Records* button.	All records will be displayed, and the combo-boxes will be cleared.	As expected.
Attempt to update the *Record Number*.	An update will not be allowed.	The *Record Number* could be updated, so I changed the field property.
Attempt to update the *Record Title*.	An update will not be allowed.	As expected.
Attempt to update the *Label*.	An update will not be allowed.	As expected.
Click on the *Re-use* field.	If it is currently unselected it will become selected, and if it is currently selected it will become unselected.	As expected.

Click on the *Delete* button.	The system will ask for confirmation. If the selection is 'no' the confirmation box will simply close. If the selection is 'yes' that record, and any associated tracks will be deleted from the system.	As expected. As expected. As expected.
Double-Click on the *Record Title*.	The *Find Track* will be opened, displaying all tracks from the record. All search combo-boxes will be disabled. The *Find Record* form will be made invisible.	As expected. As expected. As expected.
Click on the *Exit* button.	The *Find Record* form will be closed, and you will be returned to the *Main Menu.*	As expected.

Find Track

Action	Expected Result	Actual Result
Select an option from any combo-box.	The *Display All Tracks* button will become visible.	As expected.
Select valid values from each search combo-box.	They data will be filtered accordingly.	As expected.
Click on the *Display All Tracks* button.	All records will be displayed, and the combo-boxes will be cleared.	As expected.
Attempt to update all fields on the form.	It will be prevented.	The *Mix* field could be updated, so I changed the field properties.
Click on the *Current Playlist* field.	If it is currently unselected it will become selected, and if it is currently selected it will remain selected.	As expected.
Click on the *Classic* field.	If it is currently unselected it will become selected, and the *Current Playlist* field will become unselected. If it is currently selected it will become unselected.	As expected. As expected.
Click on the *Start Set* field.	If it is currently unselected it will become selected, and if it is currently selected it will become unselected.	As expected.
Click on the *Finish Set* field.	If it is currently unselected it will become selected, and if it is currently selected it will become unselected.	As expected.
Click on the *Delete* button.	The system will ask for confirmation. If the selection is 'no' the confirmation box will close. If the selection is 'yes' that track will be deleted from the system.	As expected. As expected. As expected.

Double-Click on the *Track Title*.	The *Update Track* form will be opened, displaying the details of that track. The *Find Track* form will be made invisible.	As expected.
Double-Click on the *Mix* of two different tracks.	The details of the two tracks will be displayed at the bottom of the screen. The system will ask to confirm whether you want the two tracks saving as neighbouring tracks. If the selection is 'no' the confirmation box will close, and the area displaying the details of the two tracks will be cleared. If the selection is 'yes' the tracks will be saved as neighbouring tracks.	As expected. As expected. As expected. As expected.
Double-Click twice on the *Mix* of the same track.	An error message will produced, informing the user they have selected the same track twice.	As expected.
Attempt to define the same neighbouring relationship twice.	An error message will be produced.	The system did not detect this and tried to save it to the system, causing an error. See comments for a detailed explanation and solution.
Click on the *Exit* button.	The *Find Track* form will be closed. If you arrived from the *Find Record* form you will be returned to that form, if you arrived from the *Main Menu* you will be returned there.	As expected. As expected.

Comments
The system didn't detect when I t ried to save a neighbouring relationship to the system that already existed. This meant that the neighbouring relationship wasn't saved to the system due to key violations. I needed to detect for this so an informative message could be produced to the user. To achieve this I altered the code so it would search for the proposed neighbouring relationship, before attempting to save it to the system. If it already exists an error message is produced. This alteration also needed to be made on the *Generate Set List* form.

Update Track

Action	Expected Result	Actual Result
Enter a value that doesn't exist in a combo-box.	An error message will be produced.	As expected.
Select valid values from each combo-box.	They will be accepted.	As expected.
Select a valid *Speed*.	The speed in BPM will be displayed next to the combo-box.	This did not appear, so I implemented the feature.
Attempt to leave each field blank, and click on the *Update* button.	An error message will be produced.	As expected.
Enter valid values into each field and click on the *Update* button.	The track will be updated, and the *Update Track* form will be closed. The *Find Track* form will remain.	As expected.

Click on the *Exit* button.	The track will not be updated, and the *Update Track* form will be closed. The *Find Track* form will remain.	As expected.

Generate Set List

Action	Expected Result	Actual Result
Enter a value that doesn't exist in the *Set Type* combo-box.	An error message will be produced.	As expected.
Enter a value that doesn't exist in the *Starting Style* combo-box.	An error message will be produced.	As expected.
Select valid values from each combo-box.	They will be accepted.	As expected.
Attempt to leave each option field blank, and click on the *Generate Set* button.	An error message will be produced.	As expected.
Enter text into the *Duration* field.	An error message will be produced.	As expected.
Enter text or number into the *Gig Date* field.	An error message will be produced.	As expected.
Click on the *Include Classic Tracks* field.	If it is currently unselected it will become selected, and if it is currently selected it will become unselected.	As expected.
Enter valid values into each field and click on the *Generate Set* button.	A set will be generated. The option fields, *Generate Set* button and *Weight Settings* button will be disabled. The *Accept Set List*, *Add Track to List*, *Add Classic to List* buttons will be enabled.	As expected. As expected. As expected.
Attempt to update all fields on the form.	It will be prevented.	On the set list the *Style*, *Speed* and *Mood* fields could be updated, so I changed the field properties.
Click on the *Weight Settings* button.	The *Custom Weight Settings* form will be made visible. The *Generate Set List* form will be made invisible.	As expected. As expected.
Click on the *Define New Set* button.	The set list will be cleared. The fields and buttons will be enabled and disabled as required.	As expected. As expected.
Click on the *Add Track to List* button.	The *Find Track for Set List* form will be opened. The *Generate Set List* form will be made invisible.	As expected. As expected.
Click on the *Add Classic to List* button.	The *Find Classic for Set List* form will be opened. The *Generate Set List* form will be made invisible.	As expected. As expected.

Click on the *Insert* button without selecting a track to insert.	An error message will be produced.	As expected.
Double-Click on the *Track Title* on the neighbouring track list.	The details of the track will be displayed at the bottom of the form.	As expected.
Click on the *Insert* button after selecting a track to insert.	The track will be inserted into the set list at the correct position. The positions will be updated correctly. The track count will be updated correctly.	As expected. As expected. As expected.
Click on the *Insert as finishing track* button after selecting a track to insert.	The track will be inserted into the set list at the end of the set list. The positions will be updated correctly. The track count will be updated correctly.	As expected. As expected. As expected.
Double-Click on the *Pos* field.	The details of the track will be displayed at the bottom of the form.	As expected.
Click on the *Insert as finishing track* button after double-clicking on the *Pos* field.	An error message will be produced.	As expected.
Click on the *Insert* button after double-clicking on the *Pos* field.	The system will ask to confirm whether you want to move the track. If the selection is ‘no’ the confirmation box will close, and the track will remain where it is. If the selection is ‘yes’ the track will be removed from its current position and moved to the new position. The positions will be updated correctly.	As expected. As expected. As expected. As expected.
Click on the *Remove* button.	The system will ask to confirm whether you want the track removing from the set list. If the selection is ‘no’ the confirmation box will close, and the track will remain on the set list. If the selection is ‘yes’ the track will be removed from the set list. The positions will be updated correctly. The track count will be updated correctly.	As expected. As expected. As expected. As expected. As expected.

Double-Click on the *Neighbour* field on the neighbouring tracks list.	The track will be inserted into the set list at the correct position. The *N* box will be checked. The positions will be updated correctly. The track count will be updated correctly. The neighbouring track count will be updated correctly.	As expected. As expected. As expected. As expected.
Double-Click on the *Track Title* of two different tracks.	The details of the two tracks will be displayed at the bottom of the screen. The system will ask to confirm whether you want the two tracks saving as neighbouring tracks. If the selection is ‘no’ the confirmation box will close, and the area displaying the details of the two tracks will be cleared. If the selection is ‘yes’ the tracks will be saved as neighbouring tracks.	As expected. As expected. As expected. As expected.
Double-Click twice on the *Mix* of the same track.	An error message will produced, informing the user they have selected the same track twice.	As expected.
Attempt to define the same neighbouring relationship twice.	An error message will be produced.	As expected.
Click on the *Accept Set List* button.	The *Set List* report will be opened, displaying the correct data.	As expected.
Click on the *Exit* button.	The set list will be cleared. The *Generate Set List* and *Custom Weight Settings* forms will be closed, returning you to the *Main Menu*.	As expected. As expected.

Find Track for Set List

Action	Expected Result	Actual Result
Select an option from any combo-box.	The *Display All Tracks* button will become visible.	As expected.
Select valid values from each search combo-box.	They data will be filtered accordingly.	As expected.
Click on the *Display All Tracks* button.	All records will be displayed, and the combo-boxes will be cleared.	As expected.
Click on the *Start Set* field.	If it is currently unselected it will become selected, and if it is currently selected it will become unselected.	As expected.
Click on the *Finish Set* field.	If it is currently unselected it will become selected, and if it is currently selected it will become unselected.	As expected.

Attempt to update all fields on the form.	It will be prevented.	The *Mix* and *Popularity* fields could be updated, so I changed the field properties.
Double-Click on the *Track Title*.	The *Find Track for Set List* form will be closed, and the *Generate Set List* form will be made visible. The track details will be displayed at the bottom of the *Generate Set List* form.	As expected. As expected.
Click on the *Exit* button.	The *Find Track for Set List* form will be closed, and the *Generate Set List* form will be made visible.	As expected.

Find Classic for Set List

Action	Expected Result	Actual Result
Select an option from any combo-box.	The *Display All Tracks* button will become visible.	As expected.
Select valid values from each search combo-box.	They data will be filtered accordingly.	As expected.
Click on the *Display All Tracks* button.	All records will be displayed, and the combo-boxes will be cleared.	As expected.
Click on the *Start Set* field.	An update will not be allowed.	As expected.
Click on the *Finish Set* field.	An update will not be allowed.	As expected.
Attempt to update all fields on the form.	It will be prevented.	The *Mix* and *Popularity* fields could be updated, so I changed the field properties.
Double-Click on the *Track Title*.	The *Find Classic for Set List* form will be closed, and the *Generate Set List* form will be made visible. The track details will be displayed at the bottom of the *Generate Set List* form.	As expected. As expected.
Click on the *Exit* button.	The *Find Classic for Set List* form will be closed, and the *Generate Set List* form will be made visible.	As expected.

Custom Weight Settings

Action	Expected Result	Actual Result
Move the slider for each constraint, and click on the *Save Settings* button.	The values to the right of the sliding bars will be updated according to the new positions of the sliders.	As expected.
Click on the *Default Settings* button.	The sliding bars will return to their default positions, and the values will be set to their default values.	As expected.
Click on the *Back* button	The *Custom Weight Settings* form become invisible, and the *Generate Set List* form will be made visible.	As expected.

Display Neighbouring Tracks

Action	Expected Result	Actual Result
Attempt to update all fields on the form.	It will be prevented.	As expected.
Double-Click on the *Track Title*.	The *Neighbouring Tracks* form will be opened, displaying the correct neighbouring track. The *Display Neighbouring Tracks* form will be made invisible.	As expected. As expected.
Click on the *Exit* button.	The form will be closed, returning you to the *Main Menu*.	As expected.

Neighbouring Tracks

Action	Expected Result	Actual Result
Attempt to update all fields on the form.	It will be prevented.	As expected.
Click on the *Delete Neighbouring Relationship* button.	The system will ask to confirm whether you want the neighbouring relationship deleted. If the selection is 'no' the confirmation box will close, and the neighbouring relationship will not be deleted. If the selection is 'yes' the neighbouring relationship will be deleted from the system. The *Neighbouring Tracks* form will be closed, and the *Display Neighbouring Tracks* form will be made visible.	As expected. As expected. As expected.
Click on the *Exit* button.	The *Neighbouring Tracks* form will be closed, and the *Display Neighbouring Tracks* form will be made visible.	As expected.

Appendix J – User Acceptance Testing

	Yes	**No**
Can you record the details of a record (record number, title, and label)?		
Can you record the details of a track (the record the track appears on, title, artist, mix, side, style, BPM (beats per minute), mood, popularity, whether the track can start a set, and whether the track can finish a set)?		
Can you update the details of a track?		
Can you delete records or individual tracks from the system?		
Can you search for the details of any record or track?		
Can you generate a set list from the tracks in the system?		
Does the system use selections made by you in generating the set list?		
Does the system provide several alternative tracks to the tracks on the set list?		
Does the system allow a track on the set list to be replaced by a track on the alternative list?		
Does the system allow a track on the set list to be replaced by any track in the system?		
Is the generated set list provided in a printable form?		
Does the system have a user-friendly interface?		
Does the system include documentation on how to use the system?		