ADDENBROOKES SCREENING DATABASE

Contents

Analysis

Background to problem

The Stroke Research department in Addenbrooke's Hospital is responsible for placing patients into study groups through a process called patient screening. While the team can cope with small studies, their current system is struggling with larger studies due to limitations. The screening process is achieved by hand, relying on experience of staff to decide a person's study. However, they desperately need to organize and process patients more efficiently as the department assigned more studies than the team can handle. Because of this, deadlines are being missed. Their reliance on paper medium to store patient information hinders their ability to handle larger studies due to a number of problems. Building a specific computer program would aid the department by meeting up their processing needs and storage requirements.

Client details

The project's clients focused on 2 people who represented the rest of the team. These were the people who were interviewed and observed.

- Name – Diana Day

Role – Manager

Email – diana.day@addenbrookes.nhs.uk

- Name – Sarah Finlay

Role – Research Nurse

Email - sarah.finlay@addenbrookes.nhs.uk

Description of the current system

Gaining insight into the team's system was achieved through numerus visits to the stroke research department for observation and interviews.

The department carries out studies to improve quality of care for stroke patients. However, each study needs a specific number of consented patients. Patient screening is the process of finding these patients and matching them to a suitable study. At the moment there is no formal system, therefore process is not always done in order. However, it can be broken down into these common stages:

Table 1: Stepwise Refinement for Patient Screening

1.0 Familiarize new study
1.1 Inform team members about new study
1.1.1 Update personal booklets with study's criteria
2.0 Find suitable patient that match study's criteria
2.1 Collect patient information from available resources
2.2 Investigate whether or not patient satisfies the study's criteria
3.0 Store suitable patient who satisfy study's criteria
3.1 If patient satisfies criteria, record patient's details and the study into screening book
4.0 Ask patient's consent to join study
4.1 If the patient consents, record their decision in screening book
5.0 If the number of consented patients needed is not met, move back to step 2.0
6.0 If the number of consented patients needed is met move back to step 1.0 with a new study

From observation all study criteria are stored on their local server. Access is restricted by a user name and password from their department computers. The hardware involved in the process includes:

- Eight Desktops (HP EliteDesk 800 GI) running Windows 7 Enterprise
- HP color LaserJet printer
- Local Server (Permission was not given to see the server model)
- Screening Book
- Personal Booklets

The study criteria is a list of requirements used to determine the kind of patient a study needs. There are around a dozen studies, and so a dozen criteria. To help familiarize the staff with all the requirements, they are printed into personal booklets, given to each team member.

After a study criteria is familiarized, the team start looking for suitable patients. Usually the studies need patients suffering from stroke. Information is obtained from various sources such as the hospital's database, patient interviews as well as doctor's notes. The hospital database (known as the "Epic" system shown in *figure 1* below) is the most common source. Information is also found in physical folders, emails and on their local server. The most relevant patient details are recorded into a screening book shared by the department.

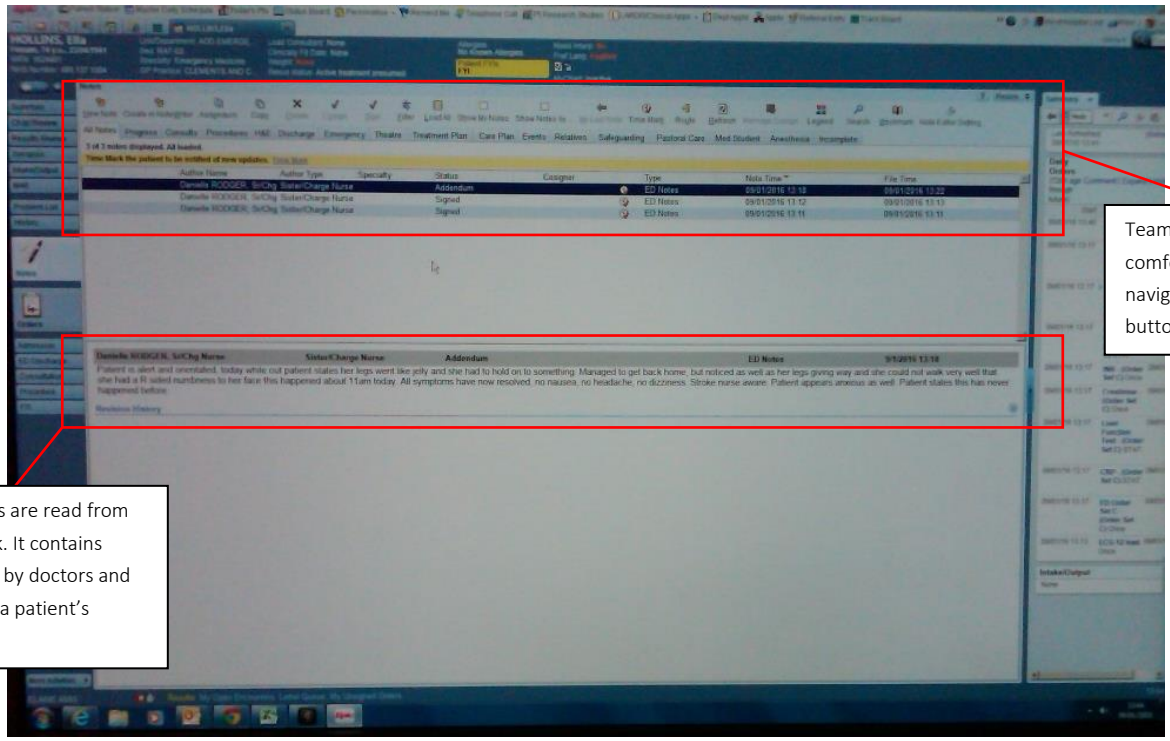


Figure 1: Addenbrooke's Database (Epic) Interface

Any potential patients are recorded into the screening book. It contains all the suitable patients, their personal details, their health conditions and their likely study group. When a suitable patient is identified, they are asked for consent to take part. Consented patients are entered into the study. The team keep looking for patients until the requirement number of patients for the study is met, while recording all their findings in the screening book.

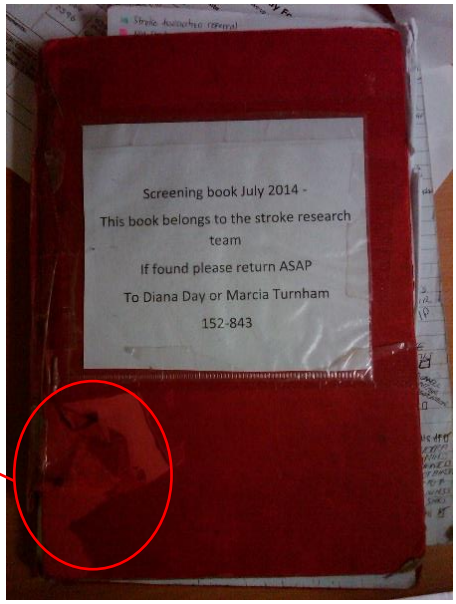


Figure 2: Screening Book used to store patient data

Identification of problems

The current system is suffering from various flaws hindering the team's effort. The most significant flaw is the use of the screening book to store their findings. From examining the book, it is difficult to read. The handwriting is very small and various sticky notes store extra info when the page runs out, making it quite unorganized. The book is also used to store data unrelated to patient screening shown in *figure 3*, such as arrival times, which adds to the confusion. It's obvious that they don't have enough space to store all their information on paper.

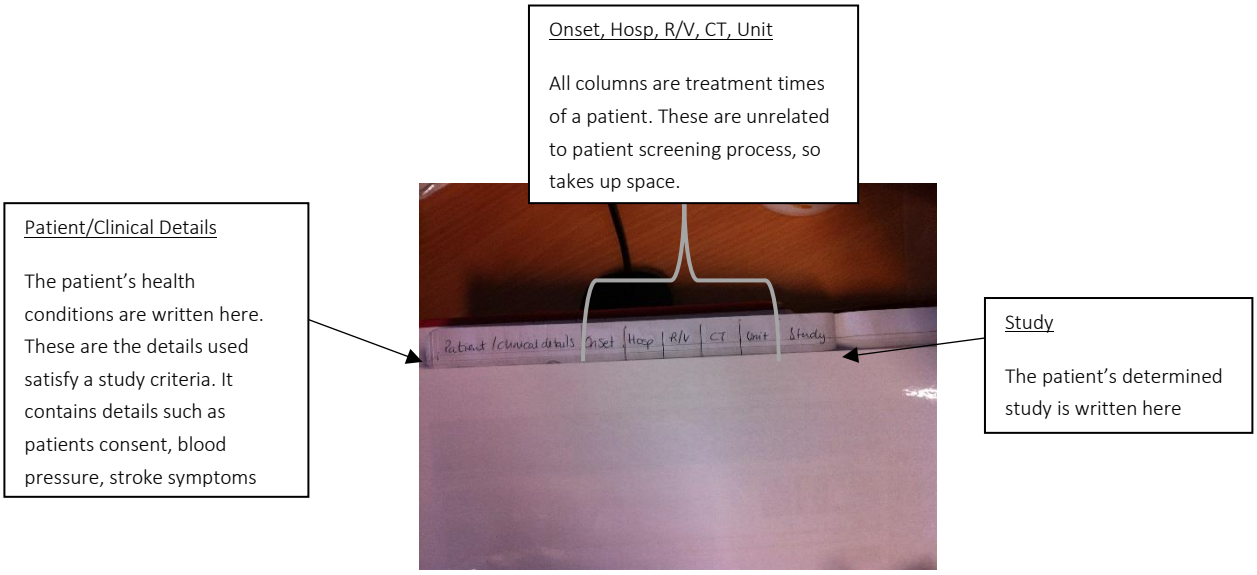


Figure 3: Contents of Screening Book (Contents are covered due to hospital policy to protect patient identity)

It's evident that the book is frequently used as its worn out and delicate to use. When it was first examined, the staff spent a few minutes searching for it around the department, so it can be misplaced easily as well.

Finding a patient is too slow. During an interview the manager Diana complained that shifting through their notes takes time. This is supported by the fact that there can be multiple sources through, including emails and books, and the fact that there are hundreds of patients to search through.

Another factor is that the team is assigned multiple studies at the same time. At the moment, there are enough studies for each team member. This is stretching their workload as some studies are too complex to handle alone. They desperately need a way to process patients quicker so that studies are finished quicker.

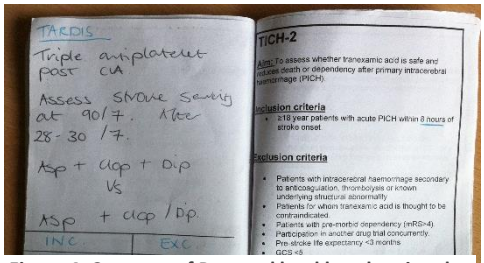


Figure 4: Contents of Personal booklet, showing the study criteria for Tardis study and TICH-2 study. The criteria is split into 2 sections: inclusion criteria and exclusion criteria.

Additionaly the personal booklets given to each team member needs consideration. They are frequently used to refer to various study criteria, but these are extremely outdated. As shown in *figure 3*, members have had to write new studies on spare pages, which is unsuitable as some criteria are too large to fit on an A5 page with normal handwriting. It was mentioned that one staff member was

responsible for replacing these booklets by printing and cutting out new ones but that it was a long process. It could be sped up if the pages were printed in a certain way. This way, cutting out pages take less time.

Identification of Prospective Users

The team is made up of 8 research nurses and one manager, all of whom are potential users as they are all responsible for collecting data into the screening book.

The client says everyone in the team are fairly computer literate, as they know how to use electronic spreadsheets and word processors in Microsoft's office suite. They are also comfortable using the computer interface for the Addenbrooke's database system. However the team is more experienced with paper documents, as they find these more comfortable and are easier to share.

As experienced nurses, including the manager, they will understand many of the medical conditions listed in the criteria and screening booklet.

Identification of user needs and acceptable limitations

A questionnaire was filled in alongside one of the clients, Sarah Finlay, during an interview to identify the user's needs.

The questionnaire states that the client would like a quicker way of storing data as well as processing patients faster. One solution would be to use a computer database, but this would not help their processing needs. The proposed system needs the ability to recommended patients to a suitable study, based on the requirements of the given criteria. This should speed up patient processing by helping choose a study quicker.

Although the program is better off on multiple computers, it would be unsuitable due to restrictions imposed by the administrator and their computer login system. If this can't be bypassed, a computer would not be able to share its database with other computers. As a result, the program will be limited to one computer. The manager has agreed to this, and she has no desire to set it up on multiple computers anyway.

The client said they had no problems with using the Addenbrooke's Epic database interface. It could be helpful if the proposed system had an interface like the Epic system as well, so learning the system can be done with ease.

Despite the team being fairly computer literate (i.e. they can work with electronic documents and access the Addenbrookes database), most members prefer paper. From observation, many of the electronic documents on their local server is printed into folders. Therefore, it is desirable that the proposed program should to be as

comfortable to use as paper format, to avoid putting of users from using it. The client states that the ability to print the contents is desirable too. The printed contents specifically refer to the study criteria, which would be used to create personal booklets.

Patient data must remain anonymous therefore upholding security needs consideration.

As mentioned earlier, the client struggles to with finding a patient in their screening book. This can be solved with some sort of filtering feature.

The client also mentions that they would like a way of identifying delays between arrival times of patients, using the treatment times in the screening book. This would be used to evaluate the team's performance. However, much of the data is unrelated to screening process and could make another project entirely. This may be avoided as this is a completely different system and could take too much time. On the other hand, it could be done as an extension if there is time.

Each nurse has different grammar therefore a criteria requirement can be written in many ways. There needs to be a way to standardize the language of health conditions to make sure are not duplicated.

Although desired by the client, the proposed system will not be on mobile devices due to time constraints but may be offered as an upgrade if the system is successful.

Summary of user needs:

- Faster storage medium
- An interface like Epic system
- Filter system built within database
- A way of processing patients faster (through automatic study recommendation)
- Ability to print documents, as most prefer paper copies

Summary of acceptable limitations:

- Proposed system is limited to one computer
- Proposed system will not be on mobile devices
- Treatment times will not be included as part of proposed system
- Users will need to agree the grammar for each health condition in the patient and criteria

Data Volumes and Analysis Data Dictionary

The project's analysis data dictionary was first drafted in rough alongside Sarah Finlay to gather data from the end user's point of view. The final analysis data dictionary was written up here in third normal form using contributing data from their screening book. There are 5 tables in total: Studies, Study Criteria, Patient Records, Patient Conditions and Health Conditions.

Studies

Field Name	*Field Type*	*Size/Range*	*Example Data*	*Purpose*	*Validation*
Study ID	Integer	8 bytes	1	Primary Key and unique identifier for study	Not blank, Is Integer and unique number
Study Name	String	50 characters	Chronos	Stores a study's name	Is String

Study Criteria

Field Name	*Field Type*	*Size/Range*	*Example Data*	*Purpose*	*Validation*
Study ID	Integer	8 bytes	1	Composite Key (part 1) and foreign key to Study ID in Studies	Not blank and is Integer
Condition ID	Integer	8 bytes	8	Composite Key (part 2) and foreign key to Condition ID in Health Conditions	Not blank and is String
Accepting Value	String	50 characters	TRUE	Represents the value needed to pass this condition	Not blank and is String

Patient Records

Field Name	*Field Type*	*Size/Range*	*Example Data*	*Purpose*	*Validation*
Patient ID	Integer	8 bytes	1	Primary Key and unique identifier for patient record	Not blank, and unique number
Forename	String	50 characters	Lorenzo	Stores first name	Is String
Middle Names	String	50 characters	Victor	Stores middle names	Is String
Surname	String	50 characters	Jumilla	Stores last name	Is String
Age	Integer	1 byte	18	Stores age	Number from 1 to 120
Gender	String	1 character	M	Stores and represents gender of the patient	Only "M" or "F" as input
Study ID	Integer	8 byte	8	Represents the study assigned to the patient. Foreign key of Study ID to Studies table	Is Integer
Consent	String	10 characters	Yes	Represents a yes or no of a patient's consent	Only "Yes", "No" or left blank as inputs

Patient Conditions

Field Name	*Field Type*	*Size/Range*	*Example Data*	*Purpose*	*Validation*
Patient ID	Integer	8 bytes	1	Composite Key (Part 1) and foreign Key to Patient ID in Patient Records	Not blank & is Integer.
Condition ID	Integer	8 bytes	2	Composite Key (Part 2) and foreign Key to Condition ID in Health Conditions	Not blank & is Integer.
Condition Value	String	50 characters	TRUE	Stores current state of condition	Not blank and is String

Health Conditions

Field Name	*Field Type*	*Size/Range*	*Example Data*	*Purpose*	*Validation*
Condition ID	Integer	8 bytes	1	Primary Key and unique identifier for health condition	Not blank, Is Integer and unique number
Condition Name	String	50 characters	Brain Hemorrhage	Stores the condition's name	Not blank and is String

Data Volumes

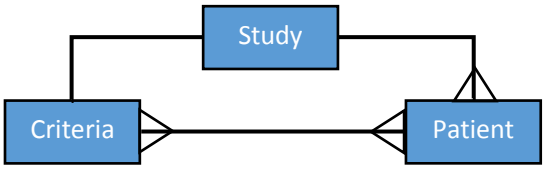
1. The system should be able to store 10 study criteria initially, with the ability to expand to 20 study criteria
2. A study should be able to store 10 patient records initially, with an expansion of 100 patient records per study
3. A patient record will contain 10 health conditions initially, with an expansion of 30 health conditions per patient record
4. The study criteria will store up to 10 health conditions initially, with the ability to expand to 50 health conditions per study

The initial system will need space for an overall of 1100 unique health conditions, 200 patient records and 10 study criteria.

The expanded system will need space for an overall of 61 000 unique health conditions, 2000 patient records and 20 study criteria.

Entity Relationship Diagram

There are 3 entities about which data is stored: study, criteria and patient. The following diagram has not been normalized yet.



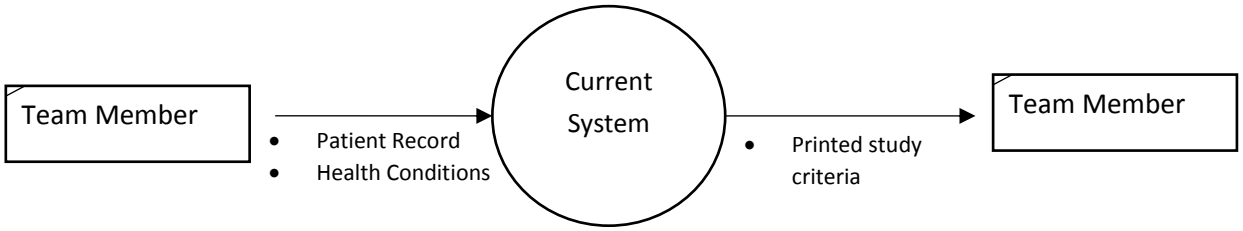
The E-R diagram above shows the following relationships:

- A patient can only be assigned one study, and a study can have many patients
- A study has one unique criteria, and a criteria belongs to one study
- Many patients can pass a criteria, and a criteria can enroll many patients

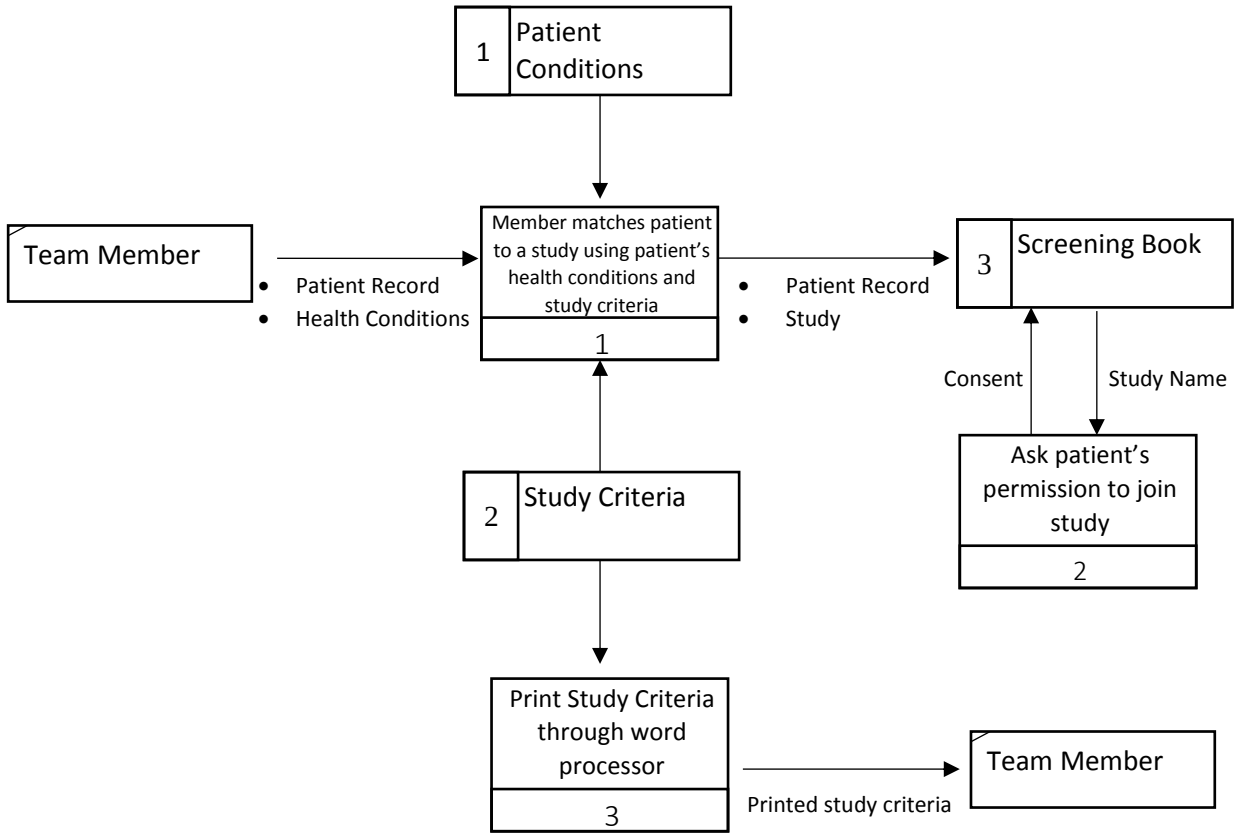
Although a patient may pass more than one criteria, a patient can only apply for one study and therefore enroll through one criteria.

Current Data Flow Diagram

Context Diagram of Current System

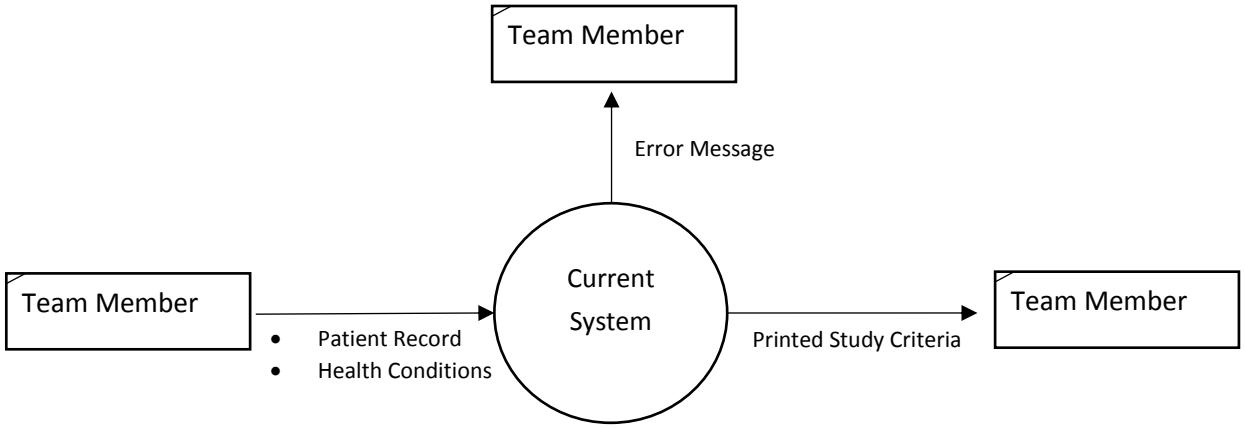


Level 1 Data Flow Diagram of Current System

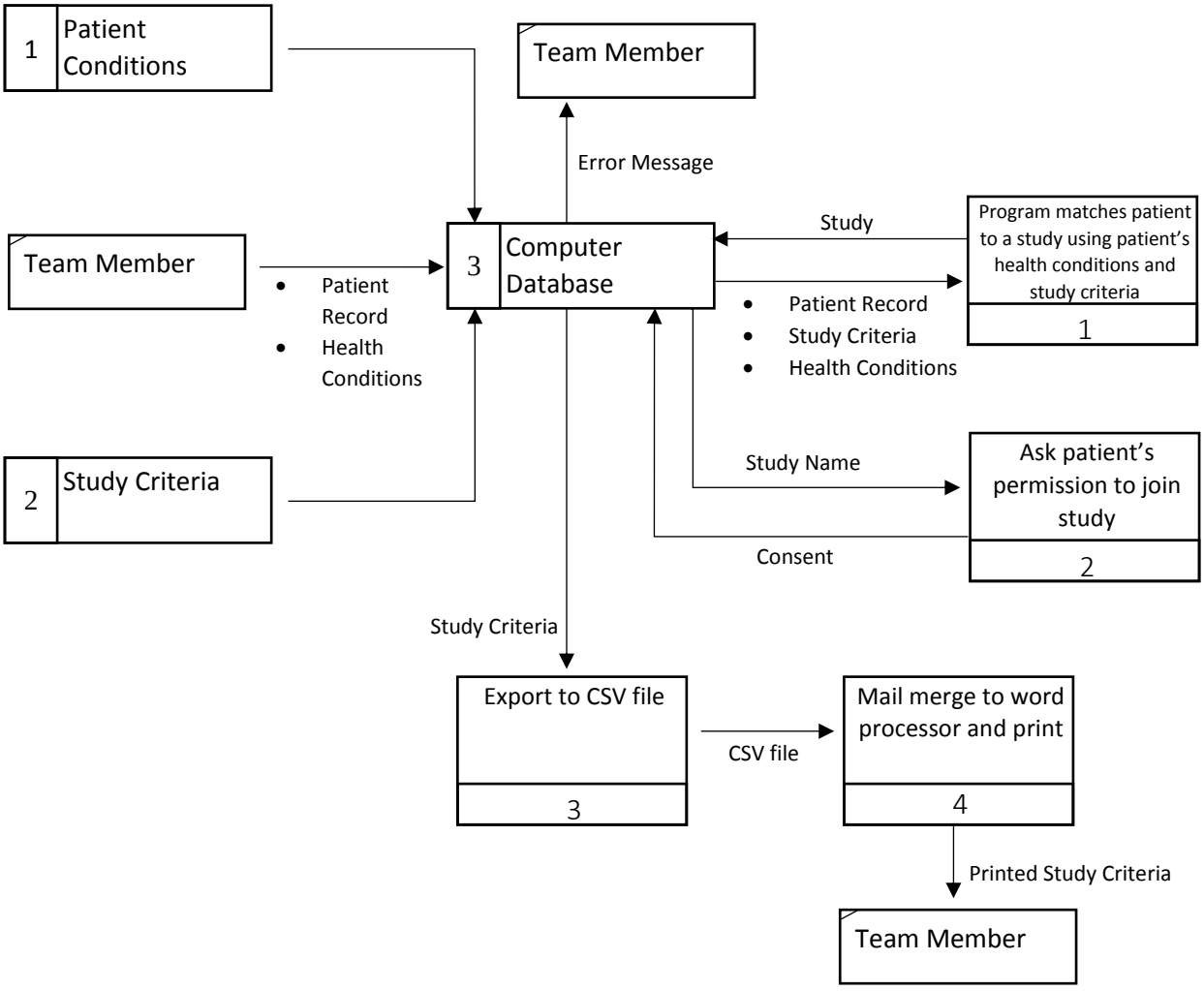


Proposed Data Flow Diagram

Context Diagram of Proposed System



Level 1 Data Flow Diagram of Proposed System



Data sources and Destinations

Data Sources & Destinations of Current System

Data Source	Process	User(s)	Output	Destination
Addenbrooke's Database	Data collection	Research Nurse	Patient details	Screening Book
Patient details	Data storage	Research Nurse	Patient is stored	Screening Book
Patient Permission	Data storage	Research Nurse	Patient's permission stored	Screening Book
Study Criteria	Data Storage	Research Nurse	Study Criteria Stored	Personal Booklet
Study	User finds patient a suitable study	Research Nurse	Patient's Study	Screening Booklet

Data Sources & Destinations of Proposed System

Data Source	Process	User(s)	Output	Destination
Addenbrooke's Database	Data collection	Research Nurse	Patient details	Computer Database
Patient details	Data storage	Research Nurse	Patient is stored	Computer Database
Patient Permission	Data storage	Research Nurse	Patient's permission stored	Computer Database
Study Criteria	Data storage	Research Nurse	Study criteria Stored	Computer Database
Study	System internally matches a patient to a study	Research Nurse	Patient's Study	Displayed onscreen
Computer Database Contents	Printing	Research Nurse	Csv file for mail merging	Paper

Overall Objectives

The overall objective is to create a database system which will allow the user to store, edit and filter patient records and study criteria, with the internal ability to automatically provide a patient's recommended study based on existing patient records and study criteria.

Feasibility of Potential Solutions

Potential Solution	Feasibility
Microsoft Access	The department's computers each have Microsoft Access installed. This office suite means the team has access to a spreadsheet and has the ability to record patients faster and to print. It's a readily available database alternative. However a relational database here would be difficult to maintain as the user would have to update each entity manually. Data would be harder to normalize as it's not normalized for them, leading to increased risk of repeating groups which is harder to maintain due to human error. Furthermore this solution would not improve patient processing as finding a patient's study would still be done manually.
C# .Net Application with csv	A .Net application programmed in C# is more flexible. The program can normalize data for the user and coded to recommended studies for potential patients. This would help maintainability and speed up patient processing. A relational database stored in a Comma Separated Values (.csv) file is straightforward to set up as it only requires a text file. Besides its readily available with mail-merging programs, potentially making it easier to create printable documents, which is one of the client's needs. The format is also widely recognized by spreadsheet programs (e.g. Excel) making it easier to export contents to other computers. However updating a relational database in this format is challenging as updates don't cascade between entities and filtering would take longer to code as it's not built in.

C# .Net Application with SQL based Database	A C# .Net application with a SQL based database eliminates some problems of a CSV file. For example the high level language can readily update and filter a database which saves development time to code and helps cascade updates in a relational database. I do have experience with setting up SQL databases, but further research is required to set it up fully (like filtering), which may take time. Unlike CSV, printing an SQL Database is not a straightforward. However, it is possible to export the contents of an SQL database into a CSV format making printing possible.

Justification of chosen Solution

Although using the existing Microsoft access as a solution is free, it would not speed up patient processing as selecting a patient's study would still be done manually. Changes made by the user would not cascade across tables too, meaning the user would have to edit each related table manually. This also risks duplicating data through human error.

Developing a .Net Application with CSV file would eliminate some the problems of problems of an office suit, such as normalization, but would still suffer from the lack of cascading updates across related tables. Filtering would have to be coded too, which may take up development time.

After evaluating the projects findings and consulting with Diana and Sarah, it can be concluded that the best solution would be to create a .Net Application with a local SQL database. This would meet their processing requirements and storage requirements as it can be programmed to automatically select a patient's suitable study as well as normalize their data. Moreover, SQL can automatically cascade updates across the relational database, reducing the need to modify each table individually. SQL language is readily capable of filtering a database, so it would save development time. Additionally, it's possible to export the database contents to a csv file making printing through mail merging possible, which is one of the client's needs.

Design

Overall System Design

The user is taken to the patient's section where they can view patient records and view patient's health conditions. From here users can add or edit patient records and health conditions. Automated study recommendation also takes place here which would give a patient's suitable studies.

Further navigation will take them to the studies section is where the user can add new studies or edit exiting ones. The option to export and print the study criteria would be available here

The main objectives are to satisfy the client's storage and processing requirements. The general solution would be to replace the screening booklet with a computer database programmed in C#, allowing the research team to store, edit, view and filter patient records and study criteria much faster. The proposed inputs would be patient records, health conditions, consent and the studies. In return the main outputs would be to display this data onscreen, a printed study criteria (preferably on A5 paper), and data storage.

The processing requirements would be satisfied by implementing a suitable algorithm which would match a patient to their suitable studies automatically. This algorithm would use the proposed database as the input with a returning an output of the patient's study.

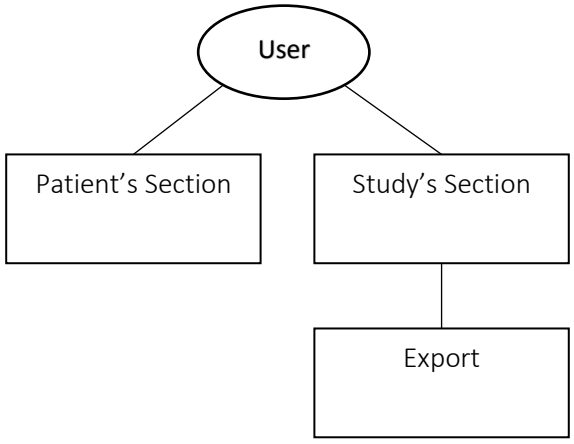
Inputs	Processes	Outputs	Storage

• Patient Records
• Health Conditions
• Studies
• Patient Consent
	• Data to database storage
	• Data to displayed format
	• Filter by ascending or descending order
	• Database updating
	• Study recommendation algorithm
	• Database to csv format
	• Mail merge printing
		• Displayed data
		• Recommended study for patient
		• Printed study criteria
		• CSV format
			• CSV file
			• Paper
			• Database Tables:
			  - Studies
			  - Study Criteria
			  - Patient Records
			  - Patient Conditions
			  - Health Conditions

Modular Structure of the System

Navigation of Forms

The user interface would be made up of 4 separate forms. Navigation between them is preferably done with buttons and drop down menus.

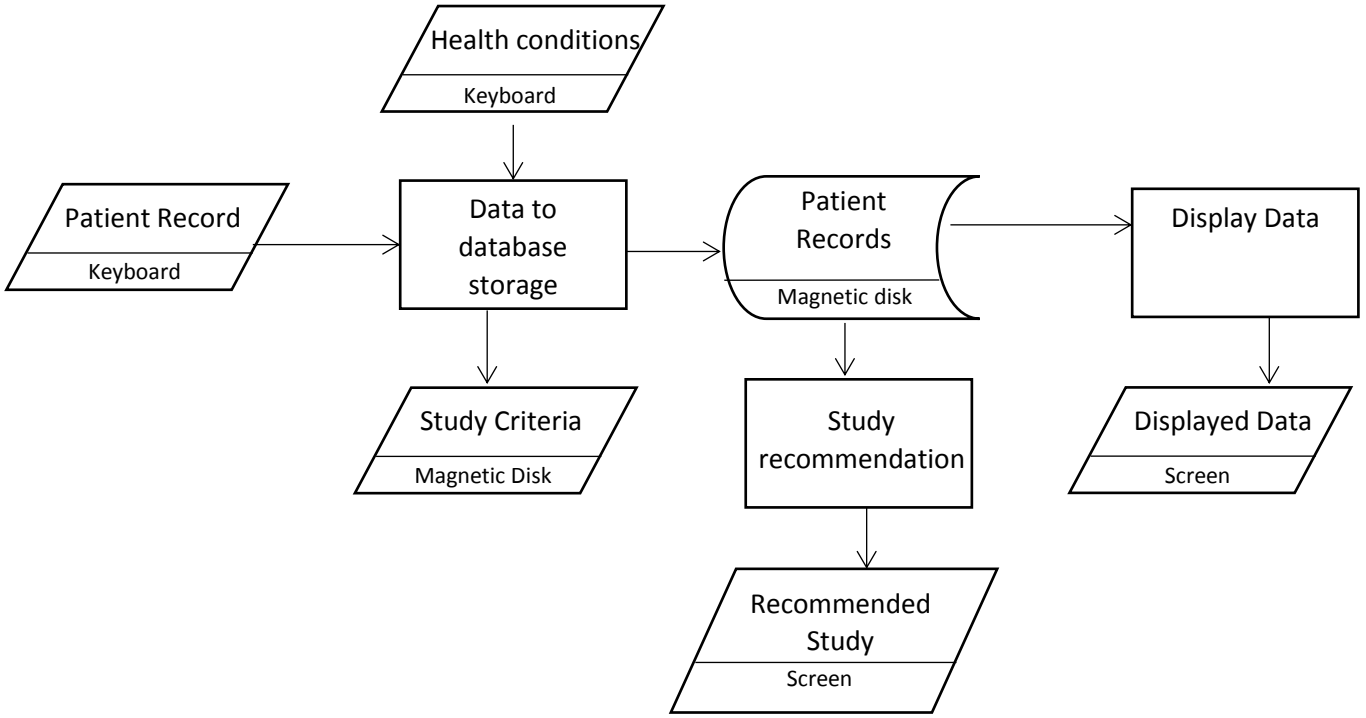


Structure Chart 1: Order of Navigation

Patient's Section Form

The user can input patient records and their health conditions into the database from the patient's section. These are stored on the database and then displayed in tables that the user can manipulate.

The study recommendation process requires input from patient records, patient health conditions and study criteria health conditions. These are compared to output a recommended study. This is displayed onscreen, but is not stored on the database automatically. There's the possibility that the computer can get it wrong, so the user should choose whether to store the recommended study. Besides, their clinical experience is more valuable than the program's decision.

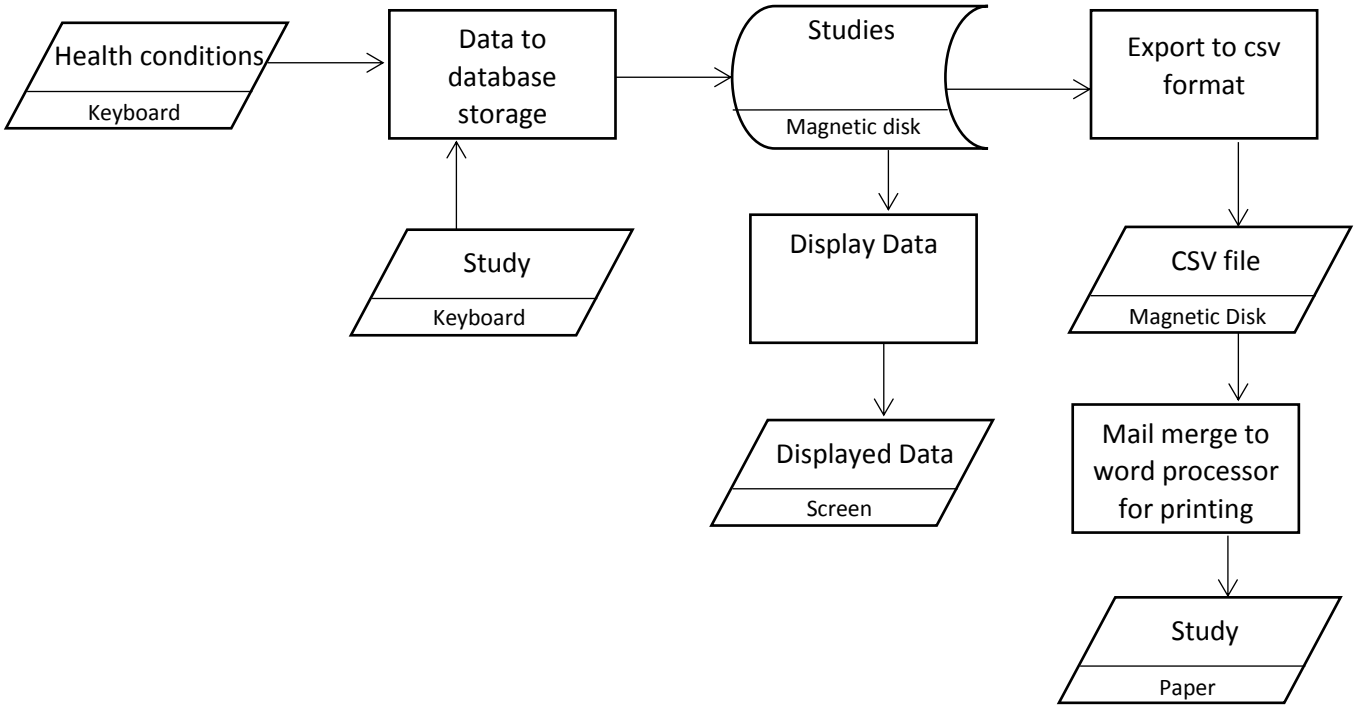


System Flowchart 1: Patient's Section

Study's Section Form

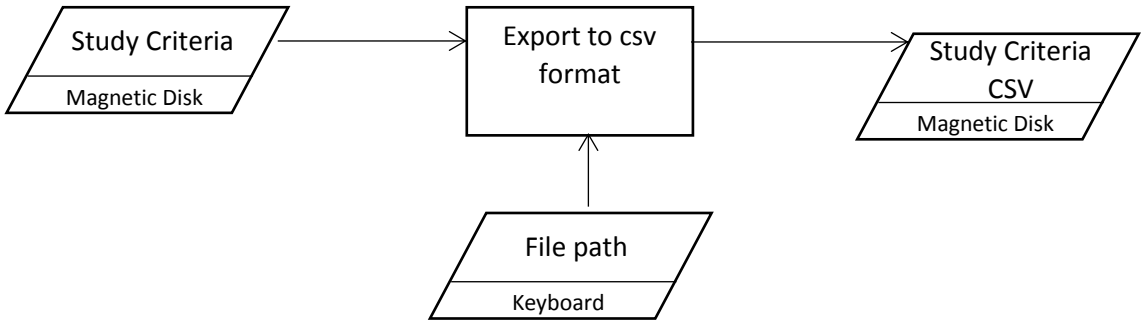
The study's section is where the user can input the study and heath condition requirements. These inputs are stored on the database, where it's then displayed in tables that the user can manipulate.

If the user wants a printed study criteria, the contents are first exported into a csv format and then mail merged onto a word processor. From here, the user can make changes in the external program.



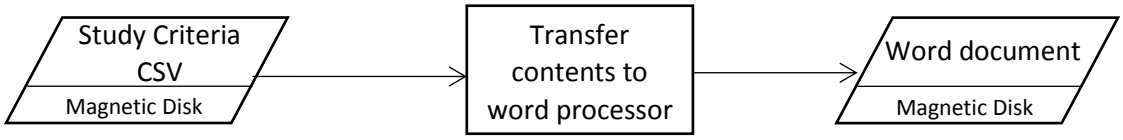
System Flowchart 2: Study's Section

Export Form



System Flowchart 3: Export

Mail Merge Form



System Flowchart 4: Mail Merge

Definition of Data Requirements

The design data dictionary is almost identical to the analysis data dictionary.

Studies

Field Name	*Field Type*	*Size/Range*	*Example Data*	*Purpose*	*Validation*
StudyID	Integer	8 byte	1	Primary Key and unique identifier for study	Not blank, Is Integer and unique number
StudyName	String	50 characters	Chronos	Stores a study's name	Is String

Health Conditions

Field Name	*Field Type*	*Size/Range*	*Example Data*	*Purpose*	*Validation*
ConditionID	Integer	8 bytes	1	Primary Key and unique identifier for health condition	Not blank, Is Integer and unique number
ConditionName	String	50 characters	Brain Hemorrhage	Stores the condition's name	Not blank and is String

Study Criteria

Field Name	*Field Type*	*Size/Range*	*Example Data*	*Purpose*	*Validation*
StudyID	Integer	8 byte	1	Composite Key (part 1) and foreign key to Study ID in Studies	Not blank and is Integer
ConditionID	Integer	8 bytes	8	Composite Key (part 2) and foreign key to Condition ID in Health Conditions	Not blank and is String
Accepting Value	String	50 characters	TRUE	Represents the value needed to pass this condition	Not blank and is String

Patient Records

Field Name	*Field Type*	*Size/Range*	*Example Data*	*Purpose*	*Validation*
PatientID	Integer	8 bytes	1	Primary Key and unique identifier for patient record	Not blank, and unique number
Forename	String	50 characters	Lorenzo	Stores first name	Is String & within 50 characters
MiddleNames	String	50 characters	Victor	Stores middle names	Is String & within 50 characters
Surname	String	50 characters	Jumilla	Stores last name	Is String
Age	Integer	1 byte	18	Stores age	Integer from 1 to 120
Gender	String	1 character	M	Stores and represents gender of the patient	Only "M" or "F" as input
StudyID	Integer	8 byte	8	Represents the study assigned to the patient. Foreign key of Study ID to Studies table	Is Integer
Consent	String	10 characters	Yes	Represents a yes or no of a patient's consent	Only "Yes", "No" or left blank as inputs

Patient Conditions

Field Name	*Field Type*	*Size/Range*	*Example Data*	*Purpose*	*Validation*
PatientID	Integer	8 bytes	1	Composite Key (Part 1) and foreign Key to Patient ID in Patient Records	Not blank & is Integer.
ConditionID	Integer	8 bytes	2	Composite Key (Part 2) and foreign Key to Condition ID in Health Conditions	Not blank & is Integer.
ConditionValue	String	50 characters	TRUE	Stores current state of condition	Not blank and is String

Definition of Record Structure

Each entity will be stored on an SQL table in a record structure:

- Studies (StudyID, StudyName)
- HealthConditions (ConditionID, ConditionName)
- PatientRecords (PatientID, Forename, MiddleNames, Surname, Age, Gender, StudyID, Consent)
- StudyCriteria (StudyID, ConditionID, AcceptingValue)
- PatientConditions (PatientID, ConditionID, ConditionValue)

StudyCriteria and Patient Conditions both have a composite key made up of foreign keys. PatientRecords also has a foreign key, under StudyID.

Validation Requirements

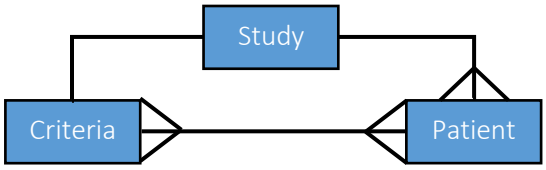
To ensure the user's data input is reasonable, validation checks take place which include range checks, type checks, length checks and lookup checks. The following tables show validation checks of all the inputs available to the user.

Field Names	*Validation Checks*	*Description*	*Error Message*
StudyID/ PatientID/ ConditionID	Datatype Integer, unique number and present	Only allow integers, must not be blank and is a unique number	ID number should be a unique number
StudyID & ConditionID	Datatype Integer, unique number combination & both fields present	Neither part of composite key must be blank, composite key must be unique and is Integer	StudyID & ConditonID should make a unique number combination
PatientID & ConditionID	Datatype Integer, unique number combination & both fields present	Neither part of composite key must be blank, composite key must be unique and is Integer	PatientID & ConditonID should make a unique number combination
Forename/ MiddleNames/ Surname/ StudyName/ ConditionName/ AcceptingValue	Datatype String & < 50 characters	Input should be under 50 characters long	No. of characters should not exceed 50
Consent	Lookup list ("Yes", "No", blank)	Only allow inputs from lookup list	Consent should be 'Yes', 'No' or left blank
Age	0 < Age <= 120	Allow ages from 0 to 120.	Age should be a number between 1 to 120
Gender	Lookup list ("M", "F")	Only allow inputs from lookup list. M means Male and F means Female	Gender should be either "M" or "F"
StudyID/ PatientID/ ConditionID	0 <= ID <= 9999999	ID should not exceed 7 digits long	ID number should not exceed 7 digits

The error messages has been adapted to use words that the team would understand by avoiding the technical terms used in computing.

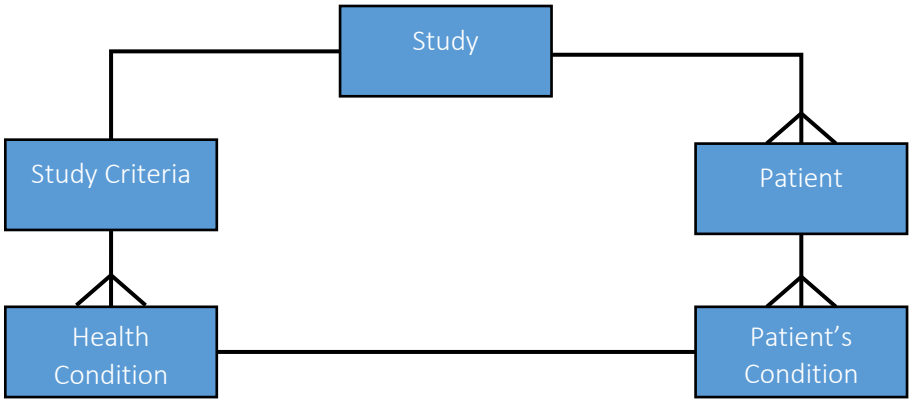
Database Design Including Normalized Relations

E-R Diagram 1 represents a non-normalized relationships of the system, showing a many-to-many relationship between the study criteria and the patient.



E-R Diagram 1: Not Normalized

E-R Diagram 2 is a revised version of *E-R Diagram 1*, showing fully normalized relationships between the database entities. There are 5 entities in total. The diagram resolves the many-to-many relationship into one-to-many relationships.



E-R Diagram 2: Fully Normalized

Description of relationships:

- A study can have many patients, and a patient can only be in one study
- A patient can have many patient conditions, and a patient's condition does not repeat for a patient
- There is one health condition for one patient condition
- A study criteria can have many health conditions, and a study criteria does not repeat a health condition
- There's one study criteria per study, and vice versa

File Organization and Processing

The export process uses a .csv file to store the study criteria. The criteria would have to be processed further to fit into one table, since its data is split between 3 tables in the database (Studies, StudyCriteria and HealthConditions), and because the csv format is best used to store a single table. The csv file will use the following format:

StudyID, ConditionID, StudyName, ConditionName, AcceptingValue — Column Names

Value 1, Value 2, Value 3, Value 4, Value 5 — First Row

Value 6, Value 7, Value 8, Value 9, Value 10 — Second Row

… — X Rows

The csv format above would be produced by combining the following record structures:

- Studies (StudyID, StudyName)
- HealthConditions (ConditionID, ConditionName)
- StudyCriteria (StudyID, ConditionID, AcceptingValue)

Queries Using SQL

SQL would be used to manipulate the database.

Return CSV contents

The SQL statement below is used for the export process to return contents from StudyID, StudyName, ConditionID and the AcceptingValue. It combines data from Studies, HealthConditions and StudyCriteria tables. The returned data is processed into a csv format.

```
SELECT Studies.StudyID, Studies.StudyName, HealthConditions.ConditionID, HealthConditions.ConditionName,
StudyCriteria.AcceptingValue
FROM Studies, HealthConditions, StudyCriteria
WHERE Studies.StudyID = StudyCriteria.StudyID
AND HealthConditions.ConditionID = StudyCriteria.ConditionID
ORDER BY StudyID, ConditionID ASC
```

Name Search

This statement returns a list of patient records that match a search term x. This is used for when the user does a name search.

```
SELECT * From PatientRecords
WHERE Forename = x OR MiddleNames = x OR Surname = x
```

Display Table

Occurs to help display the contents of a table onto the form.

```
SELECT * From (Table Name)
```

Check ID number is unique

Helps check if a user's input ID number is unique for the patient record. When nothing is returned, the ID is unique. This statement is similar for checking the uniqueness of the ID in Studies.

```
SELECT PatientID
FROM PatientRecords
WHERE PatientID = x
```

Add new Patient Record

When the user adds a new patient record, this statement is run. Consent is left blank by default as this is decided afterwards.

```
INSERT INTO PatientRecords (PatientID, Forename, MiddleNames, Surname, Age, Gender, StudyID)
VALUES (x,x,x,x,x,x)
```

Update entity with a primary key

This updates the attribute with the user's new value (x) for an entity with a primary key. This occurs after the user changes a value in PatientRecords or Studies (which both have a primary key).

```
UPDATE (Table Name)
SET (Attribute) = x
WHERE (Primary Key Attribute) = (Primary Key)
```

Update entity with a composite key

This statement runs when the user changes an attribute for an entity with a composite key.

```
UPDATE (Table Name)
SET (Attribute) = x
WHERE (Composite Key 1 Attribute) = (Composite Key 1)
AND (Composite Key 2 Attribute) = (Composite Key 2)
```

Identification of Appropriate Storage Media

The program would have to be installed onto the client's computer through 2GB USB flash drive. The program would be moved onto the USB drive then transferred onto the client's computer. A USB is used since it's an asset that's readily available for the project and it's easy to move around.

The department's standalone computers all have hard disks, a visual display unit, a keyboard and a mouse, all of which are needed to use the program. Likewise, any of them would be suitable for a large database, as the hard disks have great storage capacity and suitable transfer speeds needed to meet the client's storage requirements.

The user would start the program through an executable on their computer, however the SQL database would have to be set up manually by downloading the appropriate software onto their network. This means the programmer would have to set it up, since the user has no experience of installing SQL databases. On the other hand, should be appropriate as it will only be installed once.

As mentioned in the Analysis section, the database would need space for a maximum of 61 000 unique health conditions, 2000 patient records and 20 study criteria. Assuming each attribute uses its full size/range, the total database would take up a maximum of 12.5 MB. A hard disk has more than enough space for this.

Unicode character = 2 bytes

20 x (8 + 2(50) + 2(50)) = 228 bytes ← Studies

2000 x (8 + 2(50) + 2(50) + 2(50) + 1 + 2 + 2(10)) = 662 000 bytes ← Patient Records

61 000 x (8 + 2(50) + 2(50)) = 12 688 000 bytes ← Heath Conditions (from patient records & studies)

Total bytes = 228 + 662 000 + 12 688 000

= 12 754 428 bytes

= 12.75 megabytes

Identification of Suitable Algorithms for Data Transformation

- Add empty record – Adds an empty record to database file

```
Function addRecord( sqlTable , input )
	lastRowIndex = sqlTableRows.rows.count
	Write row[sqlTable.count] To Database sqlTable
End Function
```

- Delete record – Removes record from database file

```
Function deleteRecord( sqlTable , columnName, input )
	X = null
	Write x, [columnName, input] To Database sqlTable
End Function
```

- Update attribute – makes changes to database file

```
Function deleteRecord( sqlTable , input , columnName, rowName)
	Write input,[ columnName, rowName] To Database sqlTable
End Function
```

- Build Table – Builds table from database file

```
Function buildTable( sqlTable )
	Rows = sqlTableRows
	Columns = sqlTableColumns
	Array table[Rows.Count]
	Array row[]
	x = 0
	y = 0

	for y = 0 to Rows.count
		for x = 0 to Columns.count
			row[x] = null;
			row[x] = read [table.row[x]] From Database sqlTable
		EndFor
		table[y] = row[]
	EndFor

	Return table
End Function
```

- Generate unique number – Creates a unique id number for primary key

```
Function uniqueNumber(Table)
	uniqueNo = 0
	for i = 0 to Table.Rows.Count
		row = table.row[i]
		if (uniqueNo < row.primaryKey)
			uniqueNo = row.primaryKey
		End if
	EndFor
	uniqueNo = uniqueNo + 1

	return uniqueNo
End Function
```

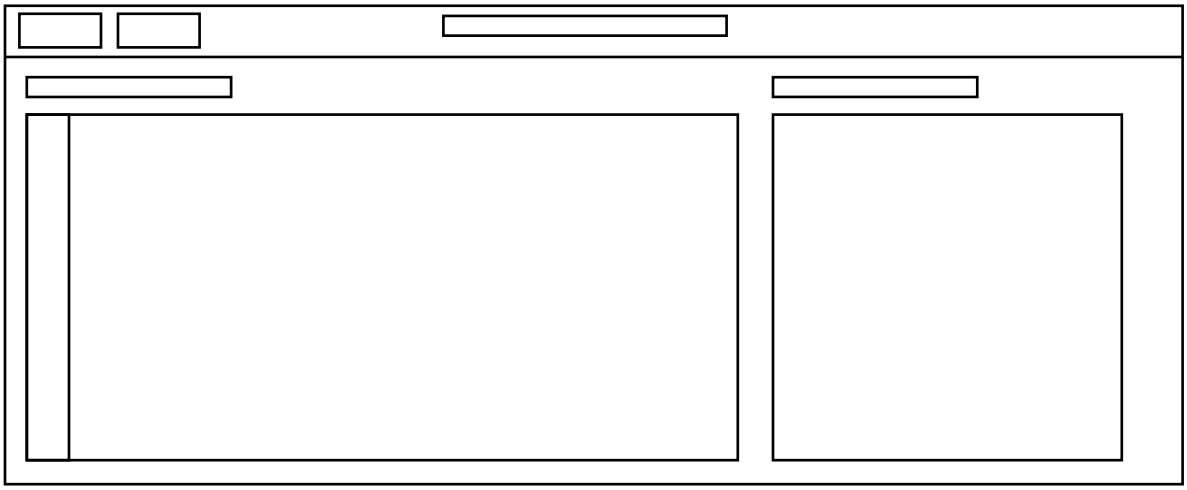
User Interface Design & HCI Rationale

The Human-Computer-Interface is designed with a colour scheme and style like the Epic user interface, by using a white background and shades of blue. This layout is meant to give a sense of familiarity, which will make the system more comfortable to use for the department team. The tables would take up much of the form's space as the user would appropriately need preview a lot of information at one time and make edits directly into the table contents. The layout feels more professional when done like this.

All the tables are to be fitted onto 2 forms, to ensure navigation is not complicated. Furthermore, all the options to export would be found on a toolbar at the top of each form, so that they can find everything in one place. The font used will be Sans Serif as it's easy to read and is the most commonly font used by the department's programs.

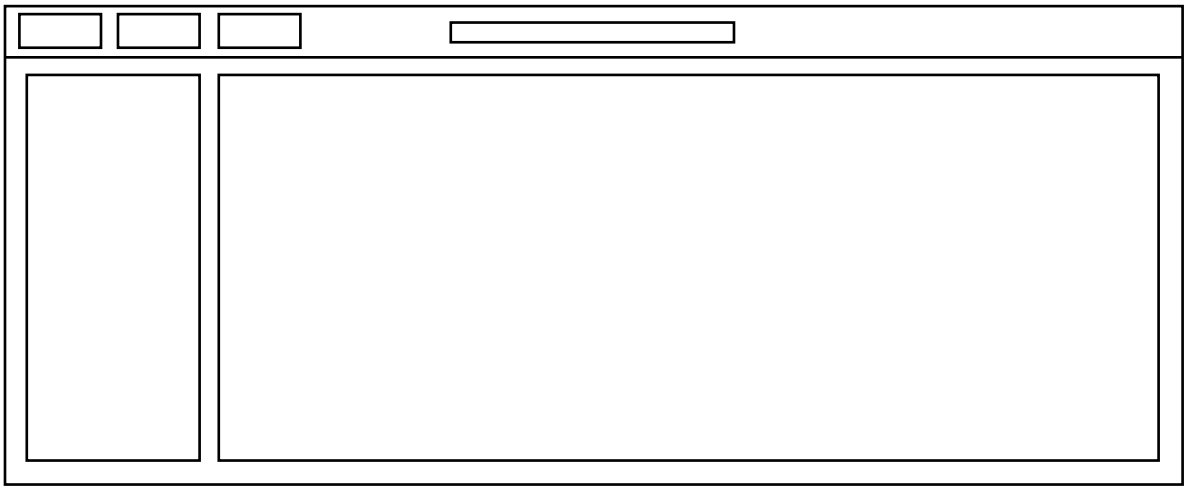
- Patient's Form

A patient's condition set would be displayed by clicking a button on their row. These buttons have to be generated while the program is running.

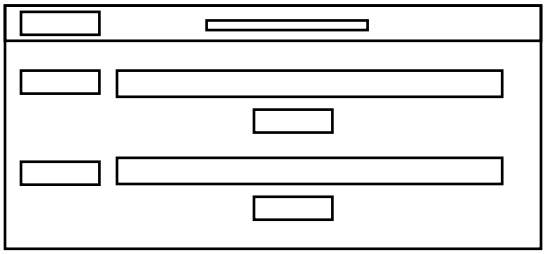


- Study Form

A study's criteria would be displayed by clicking a button on their row. These buttons have to be generated while the program is running.



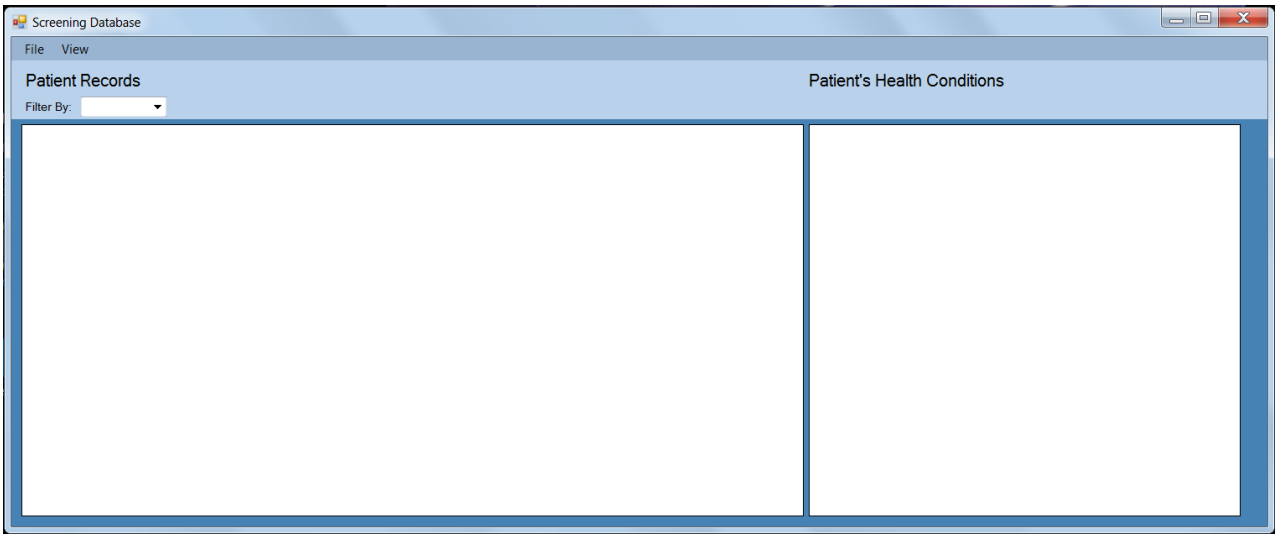
- Export/Mail Merge Form



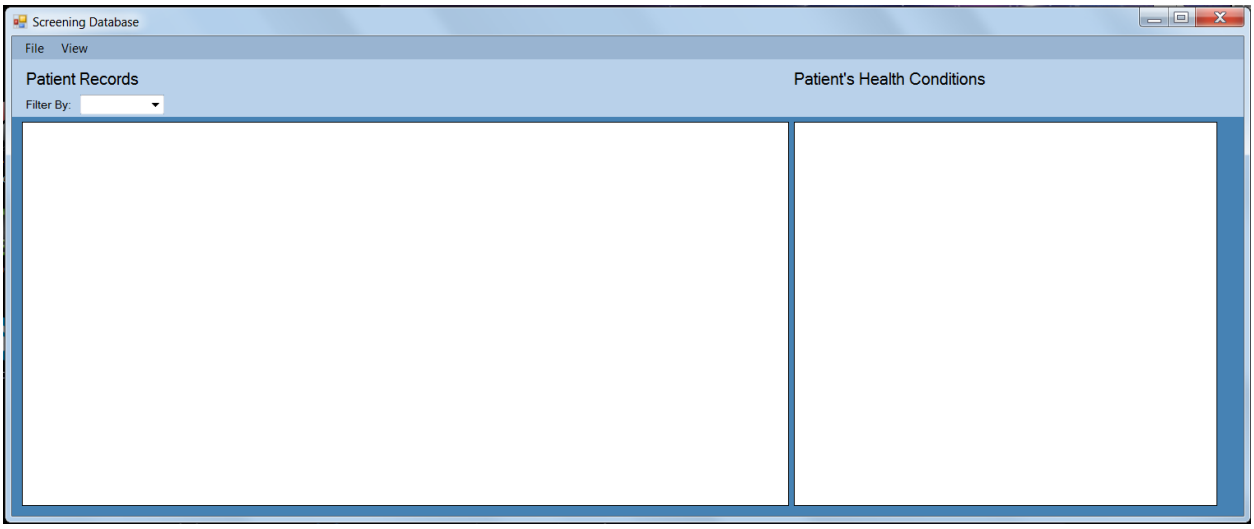
Prototype UI Designs

These prototypes were based of the planned entry designs above, created in Visual Studio 2015's design view. These are subject to small changes in the final program.

- Patient's Form



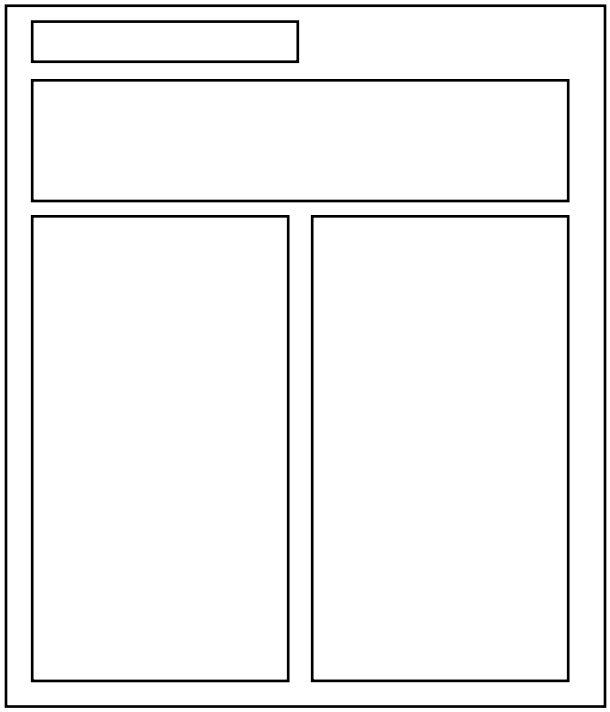
- Study Form



UI sample of planned valid output designs

- Mail merged word document

The mail merge will produce an A5 word document with a similar style to existing pages of personal booklets for consistency.



Description of measures planned for security and integrity of data

The data planned for the database will be sensitive as it would contain personal details of patients. Therefore, protecting data from unauthorized access is essential for the proposed system.

The exported csv files on the other hand would not require additional security since the study criteria is freely available to everyone.

To ensure the integrity of data is upheld, various validation checks will be implemented to ensure the user's input is reasonable. For example, the patient IDs and study IDs should have a type check to make sure the user inputs a number. Duplications are also avoided since the database entities are fully normalized, which helps reduce confusion and save space. This is further enforced with the use of ID numbers which identify repeating records uniquely.

It is important that messages are shown during an error. These should be clear enough to the point where the user corrects themselves using the message. This form of exception handling will be implemented using the try, catch, finally structure in C#, to catch any errors and interpret them into messages that the user can understand. This should prevent garbage data and uphold integrity.

Description of measures planned for system security

Another feature is the addition of the date and time access. This would represent the last time the program was accessed, providing an indicator of unauthorized access. For example, if the time of last access was after working hours, this could be evidence of unauthorized access. This should add additional security.

The use of global variables will be limited to avoid interference from other programs. Not only is this good practice but should make the system safer through further encapsulation.

Overall Test Strategy

An overall testing strategy is implemented to eliminate errors that would make the program difficult to use.

Black box is be carried out to individual functions of the program to test whether or not corresponding outputs are correct. This will involve testing button event handlers for correct behavior, such as testing the export button for the correct CSV format and will especially apply for the study recommendation algorithm, where inputting a patient should output a suitable study. The tests would also be used to test the GUI and the system's performance. The test inputs will be carefully selected as to mimic the user. This should help collect a variety of errors for correction.

White box testing will be used to specify the nature of the error in detail and help produce a correction. This will be done by breaking down the system into its fundamental parts and seeing if they are logical. This should provide a testing strategy for the study recommendation algorithm.

Preliminary Test Plan

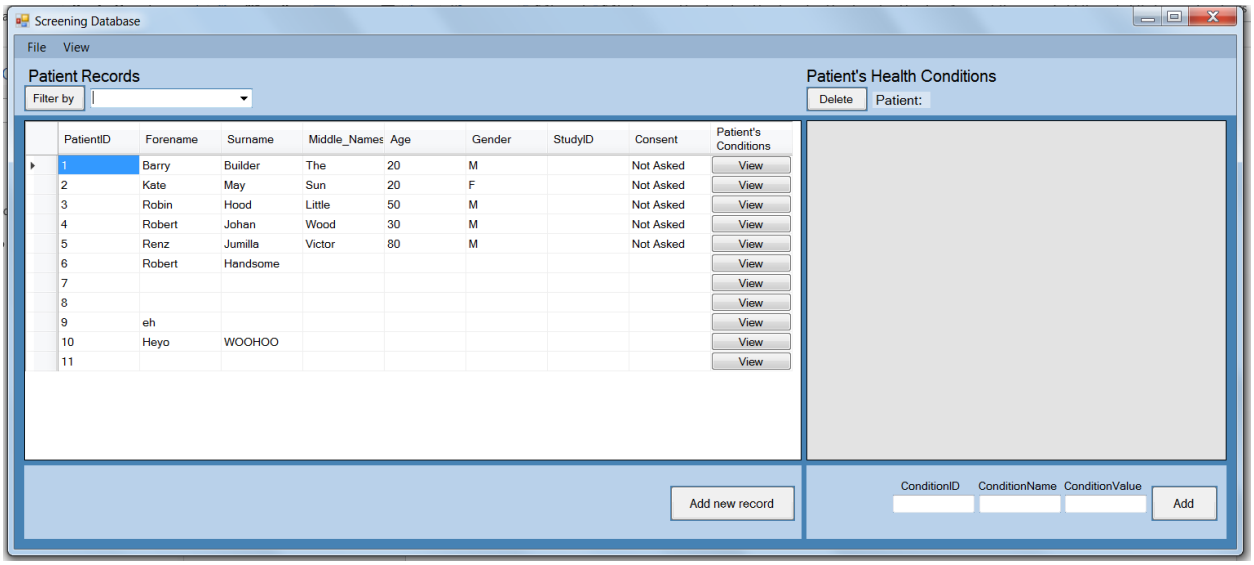
The preliminary plan lays out a systematic testing strategy for checking correct behavior of each component.

Test Series	*Purpose of test series*
1.0	Test run of navigation controls (top-down testing): do navigation buttons lead to correct forms? Does a control have the desired behavior? Does navigation flow well?
2.0	Validate changes made by user are in correct SQL tables
3.0	Correct Study recommendation behavior
4.0	Correct behavior of logic operations (if/else, case statements)
5.0	Correct csv format
6.0	Correct mail merge format

Technical Solution

The program was created using a few pre-built in classes, some of which could not be avoided. Using Visual Studio 2015 to build the solution meant that use of the DataTable class, DataGridView class and System.Data.SqlClient class were essential to create a functioning SQL database. These classes interlinked so the solution would not be possible without all of these.

Patient's Form



Patient's Form Code

```
namespace ComputingProject
{
public partial class Form1 : System.Windows.Forms.Form
{

public Form1()//Main Entry point for Patient Form
{
    InitializeComponent();

    this.mainInit();


}
//---------- Set out initial state of form at startup ----------
public void mainInit()
{
    //---------- Set up Patient Record Table ----------
    DataTable dataTablePatients = new DataTable();
    dataTablePatients.TableName = "PatientRecordsDataTable";
    mySql.refreshTable(dataTablePatients, PatientsGridView, "PatientRecords");
    mySql.syncToSQLPrimaryTable(dataTablePatients, PatientsGridView);
    myFunctions.addViewColumn(PatientsGridView, "Patient's Conditions", "View");

    PatientsGridView.CellValidating += validatePatientsData;

    //Populate filter options
    object[] filterOptions = new object[] { "PatientID", "Forename", "Surename",
"Middle_Names", "Age", "Gender", "StudyID", "Consent" };
    filterComboBox.Items.AddRange(filterOptions);

    //---------- Set up Patient Conditions Table ----------
    DataTable dataTablePatientConditions = myFunctions.patientConditions(1);
dataTablePatientConditions.TableName = "PatientConditionsDataTable";

}
```

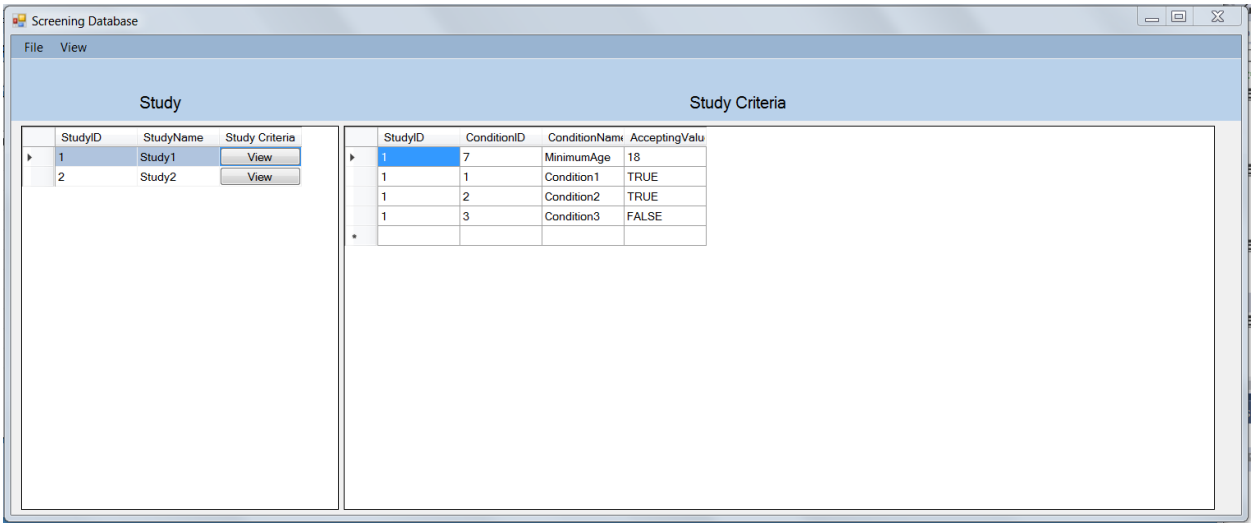
```
//-------------------------------- Event Handlers -------------------------------------

//Validate check users input on cell changed event handler
private void validatePatientsData(object sender, DataGridViewCellValidatingEventArgs
args)
{
    myFunctions.hint("Validating");

    string columnName = PatientsGridView.Columns[args.ColumnIndex].HeaderText;
    object inputValue = args.FormattedValue;

    ArrayList errorList = myFunctions.validatePatientRecord("PatientRecords", columnName,
inputValue);

    if (errorList.Count != 0)
    {
        string errorMsg = "";
        foreach (string x in errorList)
        {
            errorMsg += x;
            MessageBox.Show(errorMsg);
        }
        args.Cancel = true;
    }
}
```

PatientID	Forename	Surname	Middle_Names	Age	Gender	StudyID	Consent
1	Bob	Builder	The	20	8		Not Asked

```
//Add Empty Patient Record
private void addPatientOnButtonClick(object sender, EventArgs args)
{
    //Generate unique patientID
    int patientID = 0;
    for (int i = 0; i <= PatientsGridView.Rows.Count - 1; i++)
    {
        if (patientID < int.Parse(PatientsGridView.Rows[i].Cells[0].Value.ToString()))
        {
            patientID = int.Parse(PatientsGridView.Rows[i].Cells[0].Value.ToString());
        }
    }
    patientID += 1;

    //Call insert function from mySql class to add new row
    string[] inputRow = new string[] { patientID.ToString() };
    string[] columnNames = new string[] { "PatientID" };
    mySql.insert("PatientRecords", columnNames, inputRow);
    DataTable dataTablePatients = new DataTable();
    dataTablePatients.TableName = "PatientRecordsDataTable";

    //Refresh table interface and sync future changes to related SQL table
    mySql.refreshTable(dataTablePatients, PatientsGridView, "PatientRecords");
    mySql.syncToSQLPrimaryTable(dataTablePatients, PatientsGridView);

}
```

```
//Navigate to Study Form
private void studySectionAccess(object sender, EventArgs e)
{

    Form2 studyForm = new Form2();
    studyForm.Closed += (s, args) => this.Close();
    studyForm.Show();
}

//Close form on close button click
private void closeToolStripMenuItem_Click(object sender, EventArgs e)//Close option
{
    this.Close();
}




}
}
```

Study Form



From here the user can view existing studies and criteria. The study section is navigated to from the patient form. The user can click on the view buttons next to the rows to view its study criteria.

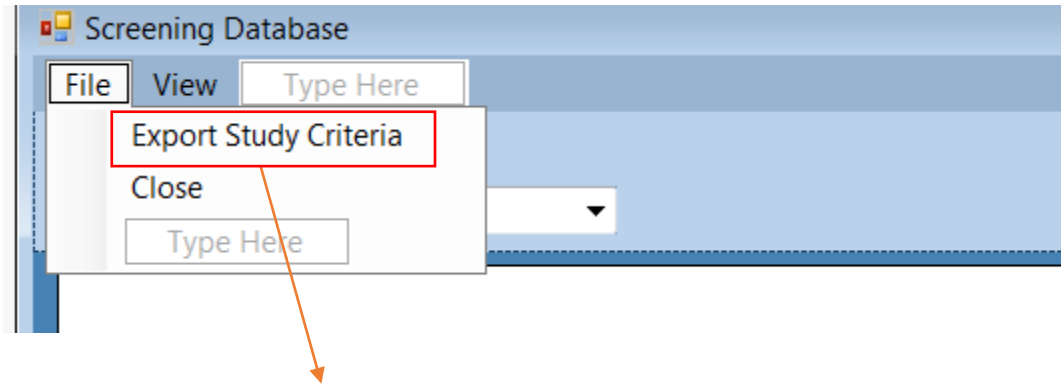
Study Form Code

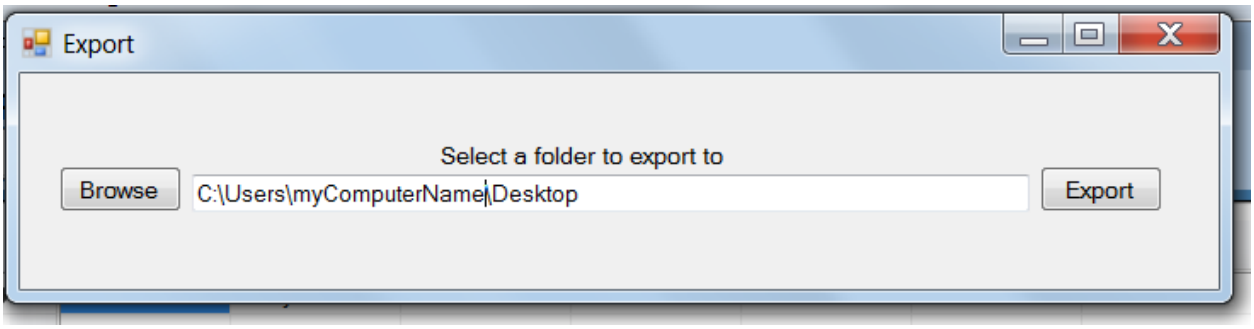
```
namespace ComputingProject
{
public partial class Form2 : Form
{
public Form2()
{
    InitializeComponent();
    this.mainInit();
}


public void mainInit()
{
    //---------- Set up study Table ----------
    DataTable dataTableStudies = new DataTable();
    dataTableStudies.TableName = "StudiesDataTable";
    mySql.refreshTable(dataTableStudies, StudyGridView, "Studies");
    mySql.syncToSQLPrimaryTable(dataTableStudies, StudyGridView);

    myFunctions.addViewColumn(StudyGridView, "Study Criteria", "View");

    myFunctions.hint(StudyGridView.ColumnCount.ToString());
    myFunctions.hint(dataTableStudies.Columns.Count.ToString());
```

```
            myFunctions.syncSQLCompositeTable(conditionsDataTable, criteriaGridView);

        }
    }

            private void openExportForm(object sender, EventArgs e)
            {
                ExportForm exportForm = new ExportForm();
                exportForm.Show();
                exportForm.table = "StudyConditions";
            }
        }
    }
```

Export Form

The export process uses the built in system.IO class to make stream writing possible. The code is run when the user clicks the export button the csv file to. The procedure then takes the user's selected folder path and writes a new csv file there.





Export Form Code

```
namespace ComputingProject
{
public partial class ExportForm : Form
{
    public ExportForm()
    {
        InitializeComponent();
    }
    private string tableToExport;//Field
    public string table//Property
    {
        get
        {
            return tableToExport;
        }
        set
        {
            tableToExport = value;
        }
    }

    private void browseButton_Click(object sender, EventArgs e)//Select a file path to
export to
    {
        FolderBrowserDialog browseDialog = new FolderBrowserDialog();
        if (browseDialog.ShowDialog() == DialogResult.OK)
        {
            filePathBox.Text = browseDialog.SelectedPath.ToString();
        }
    }

    private void ExportButton_Click(object sender, EventArgs e)//Export SQL database into
file path
    {
        myFunctions.exportCrtieriaToCsv(filePathBox.Text,table);
        this.Close();
    }
}//End Form ExportForm
}//End namespace Computing Project
```

myFunctions class

The myFunctions class stores most major functions used for the system, including the export process, validation checks and processes for dataGridview. Many event handlers called their functions from this class.

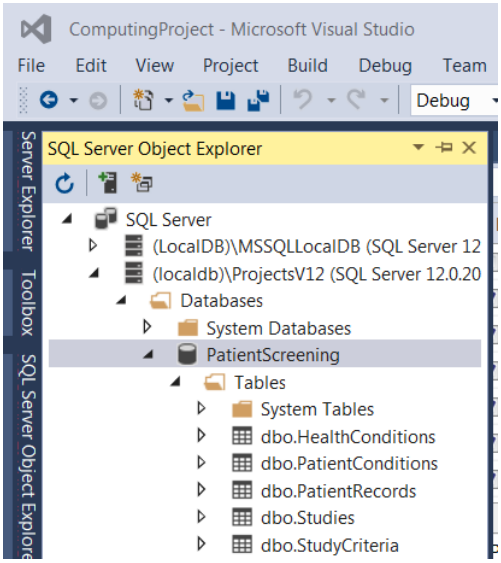
mySQL Class

This class is dedicated to functions related to SQL manipulation. Any function involving a connection to the SQL database will likely be placed here.

The SQL Database

The local database (called Patient Screening) was created using SQL Server Data Tools. This tool is an extension tool of Visual Studio 2015. To use the database from the end user's point of view, the client will have to download SQL Server Express. Although the SQL code that created the PatientScreening database was auto generated, all of the tables were coded manually.

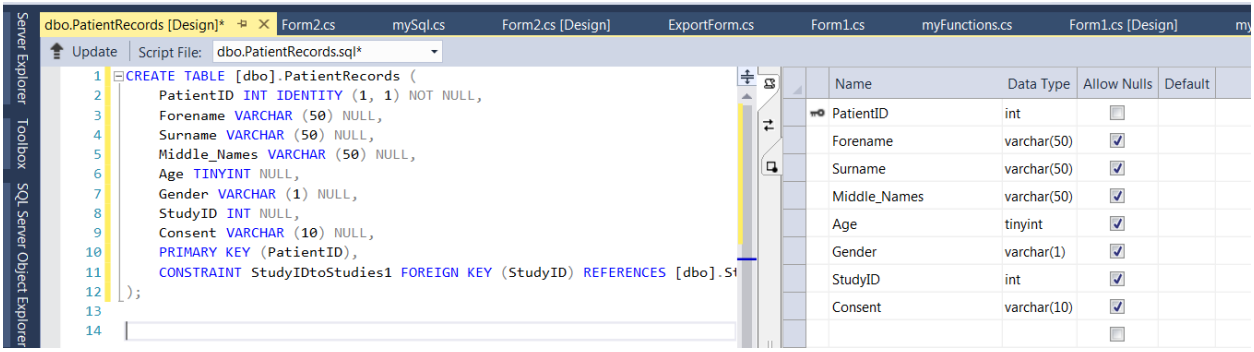
A table was created for each normalized entity. These entities are called HealthConditions, PatientConditions, PatientRecords, Studies and StudyCriteria. The picture below shows each entity under the PatientScreening database.



The following screenshots show the SQL code. The code is on the left, and the design view is on the right.

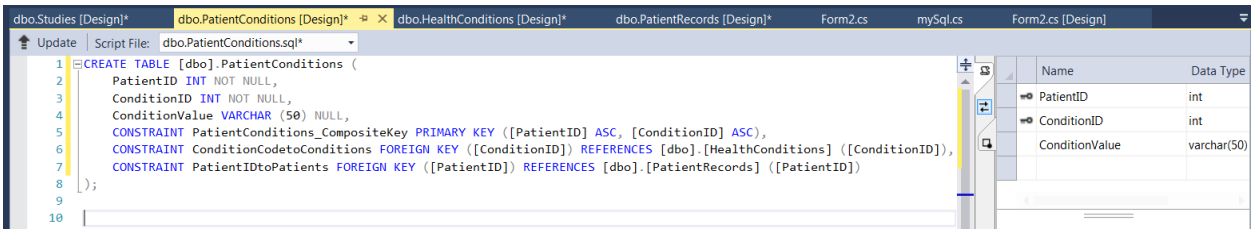
PatientRecords

Stores a patient. PatientID is the primary key and StudyID is a foreign key from the Studies table.



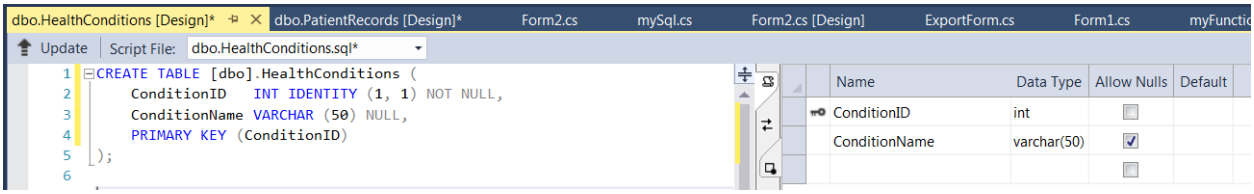
PatientConditions

Stores the patient's condtions of a patient from PatientRecords. PatientID & CondtionID make up a composite key. Both of these are foreign keys too. PatientID is a foreign key from PatientRecords and ConditionID is a foreign key from PatientConditions



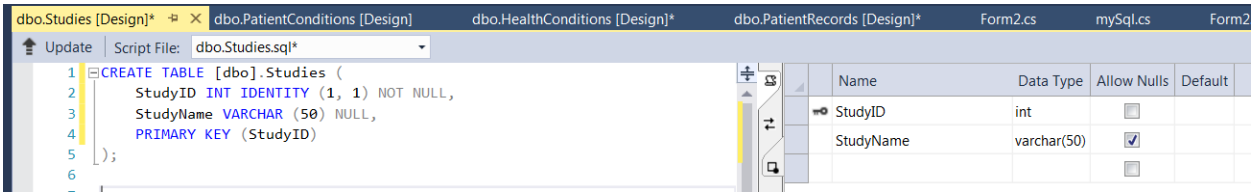
HealthConditions

Stores health conditions for the study. ConditionID is the primary key.



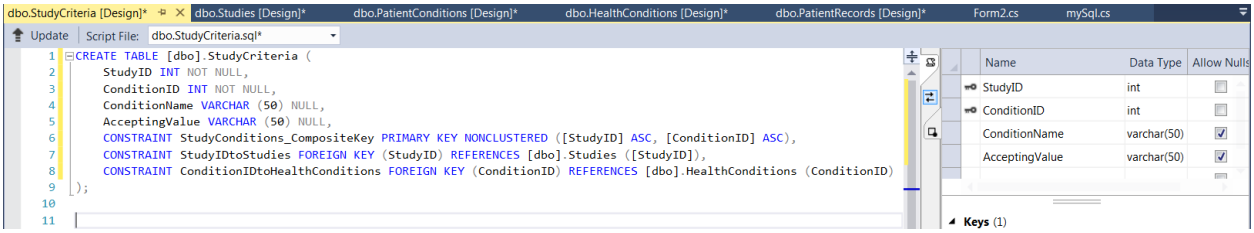
Studies

Stores studies. StudyID is the primary key.



Study Criteria

Stores the study criteria of each study. Has a composite key.



System Testing

System testing will use the preliminary test plan to carry out a detailed test plan. This will involve a variety of validation checks for a range of test data. This is to make sure program breaking errors are eliminated before use by the client.

Test Series	*Purpose of test series*
1.0	Test run of navigation controls (top-down testing): do navigation buttons lead to correct forms? Does a control have the desired behavior? Does navigation flow well?
2.0	Validate changes made by user are in correct SQL tables
3.0	Correct Study recommendation behavior
4.0	Correct behavior of logic operations (if/else, case statements)
5.0	Correct csv format
6.0	Correct mail merge format

Patient's Form

Test Number	*Test Data*	*Purpose*	*Expected Result*	*Actual Result*
1	Program loads within 5 seconds	Ensure program becomes fully functional in a reasonable time	Program starts up before 5 seconds pass	Program starts within 5 seconds

2	Input '1' in patientID of second row	Makes sure inputting duplicate id numbers are ignored	ID number should not be stored as id 1 already exists	ID is not stored on sql tables, showing an error message, but table still shows change (Screenshot 1)
3	Input 'Bob' in Forename of first patient	Check changes are stored back to sql database	Forename cell should change from Barry to Bob permanently, even after program is closed	Forename has been changed permanently, even after program restarts (Screenshot 2)
4	Click View Button of first patient	Check health conditions assigned to patient is correct	Should display correct health conditions of patient	Health conditions are displayed correctly (Screenshot 6)
5	Click "Add Records"	Check if row is added correctly to Patient Records	Empty row with a unique patientID number is added & saved to sql	Empty row added successfully and saved (Screenshot 4)
6	Click View button of second patient	See if study recommendation assigns the patient the correct study	Study1 & Study2 should be detected	Study1 & Study2 are detected (Screenshot 3)
7	Enter –20 in Age column	Check if unreasonable ages are being ignored	Patient's age should be blocked from being entered	(Screenshot 5)
8	Click Add button under patient health conditions	Check if health conditions are being added to database	New health record should be displayed	No response, nothing happens
9	Click delete button next to patient	Should remove patient record from table and sql database	Patient record should be removed from table and sql database	Program freezes, needed to restart
10	Select "Middle_Names" in filter options and click filter	Check if table gets sorted in ascending order of Middile_Names	Table should be sorted in ascending order via Middile_Name column	Table is now sorted alphabetically (screenshot 7)
11	Enter "Female" in Gender Column	Check if inputs outside cells lookup list is rejected	Change should be rejected as input should only be "M" or "F"	Change not stored, but changes are left on the table (screenshot 8)
12	Input "-1" in StudyID	Check if ID numbers that're out of range are rejected	Change should be rejected and not stored as its out of range	Screenshot 9

13	Input "6" in StudyID	Check if ID numbers that don't exist are rejected	Change should be rejected and not stored	Screenshot 10
14	Input "2"	Check if IDs within range are accepted	Change should be stored and displayed	Screenshot 11
15	Click Close	Check if program exits successfully	Program closes	Program successfully closes
16	Click View->Study Criterias	Should open study criteria form	Study form starts	Screenshot 12

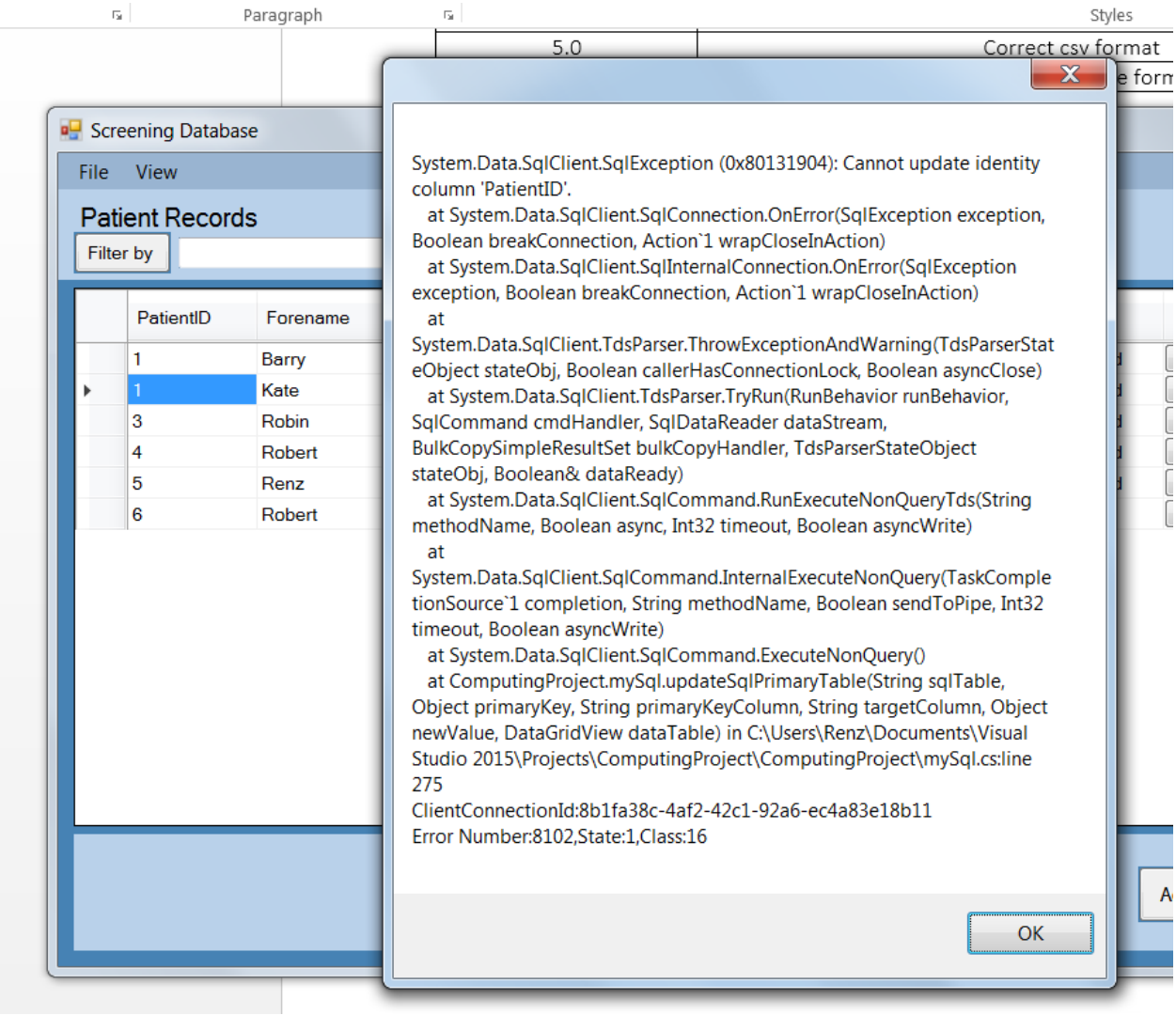
Study Form

Test Number	*Test Data*	*Purpose*	*Expected Result*	*Actual Result*
1	Click view of first study	Check if study displayed	Should display study criteria for study	Screenshot 13
2	Change name of first study into Study0	Check if new study name is stored	New study name should be stored and displayed	Screenshot 14
3	Change ID of first study to 3	Check if program accepts new studyID	StudyID should be accepted and stored	Screenshot 15
4	Click File -> Export Study Criteria	Should open export form	Export form should open	Screenshot 16

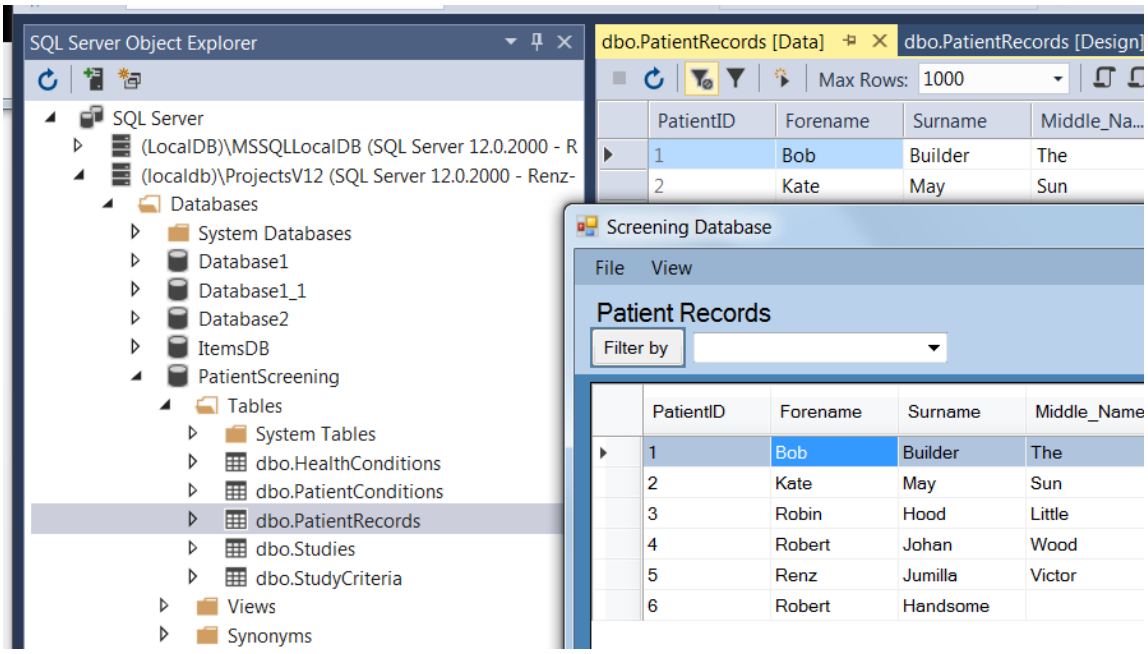
Export Form

Test Number	*Test Data*	*Purpose*	*Expected Result*	*Actual Result*
1	Click Browse	Open folder browser to get file path	Folder browser opens	Screenshot 17
2	Select file path from Folder browser	File path should be copied into central text box	File path copied into text box	Screenshot 18
3	Click export button, when file path is set to desktop	Checks to see if csv file is created in the correct format	CSV file is created on desktop in correct format	Screenshot 19

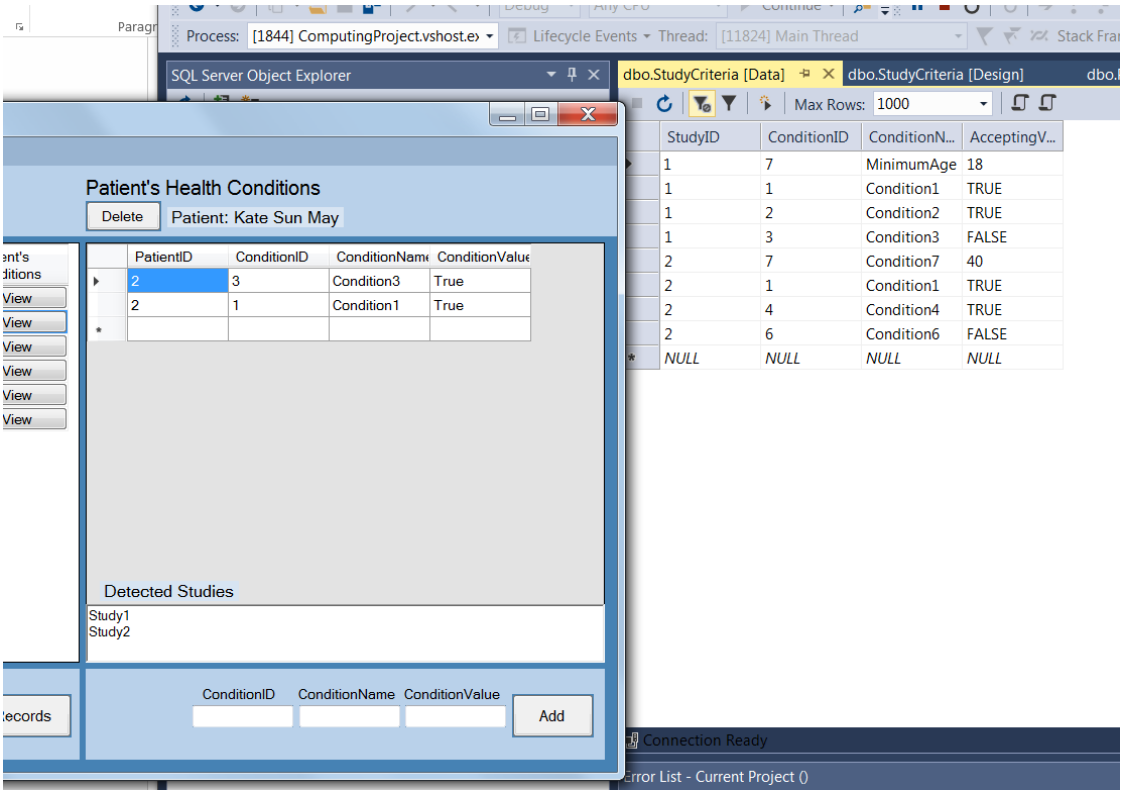
Validation Screenshots



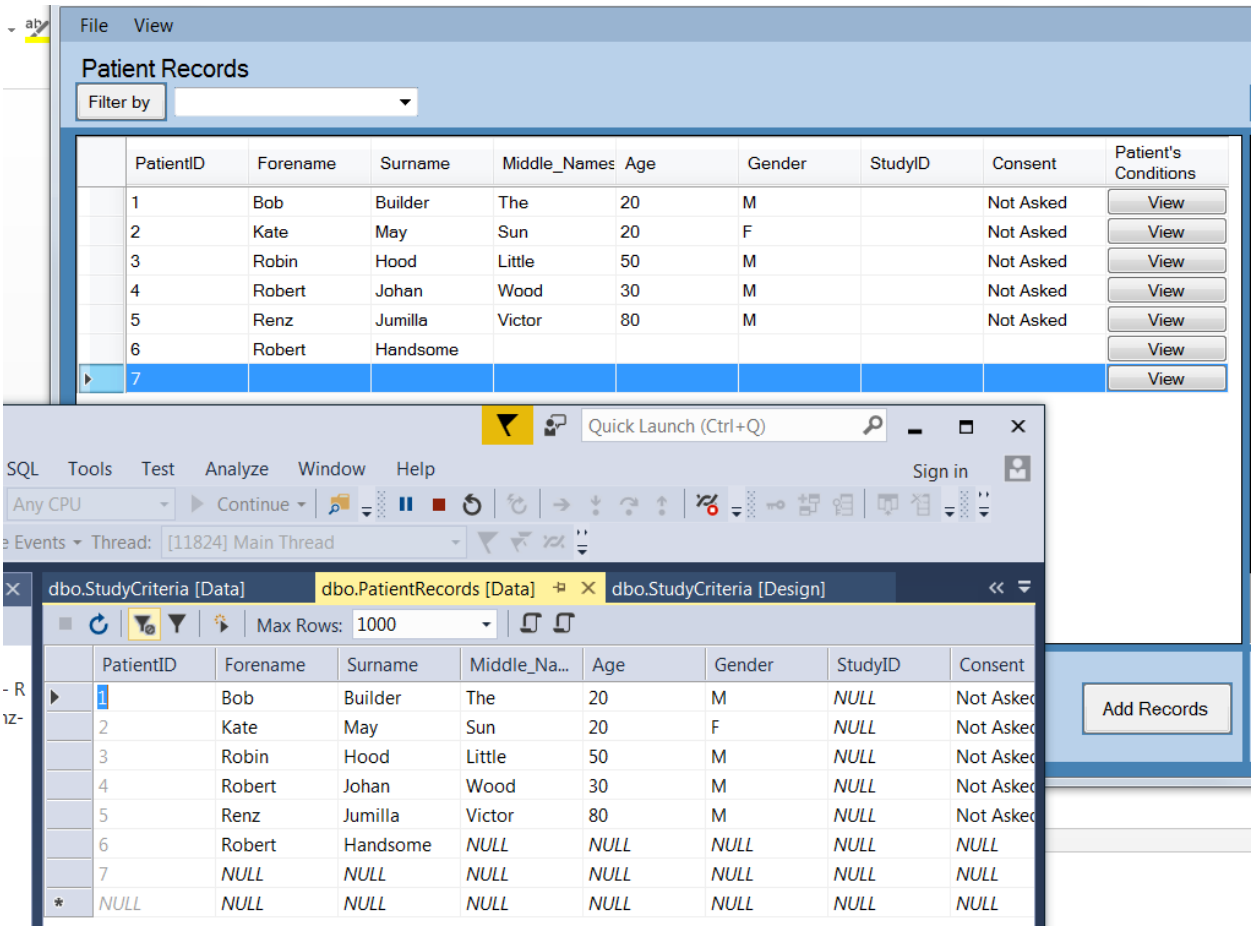
Screenshot 1



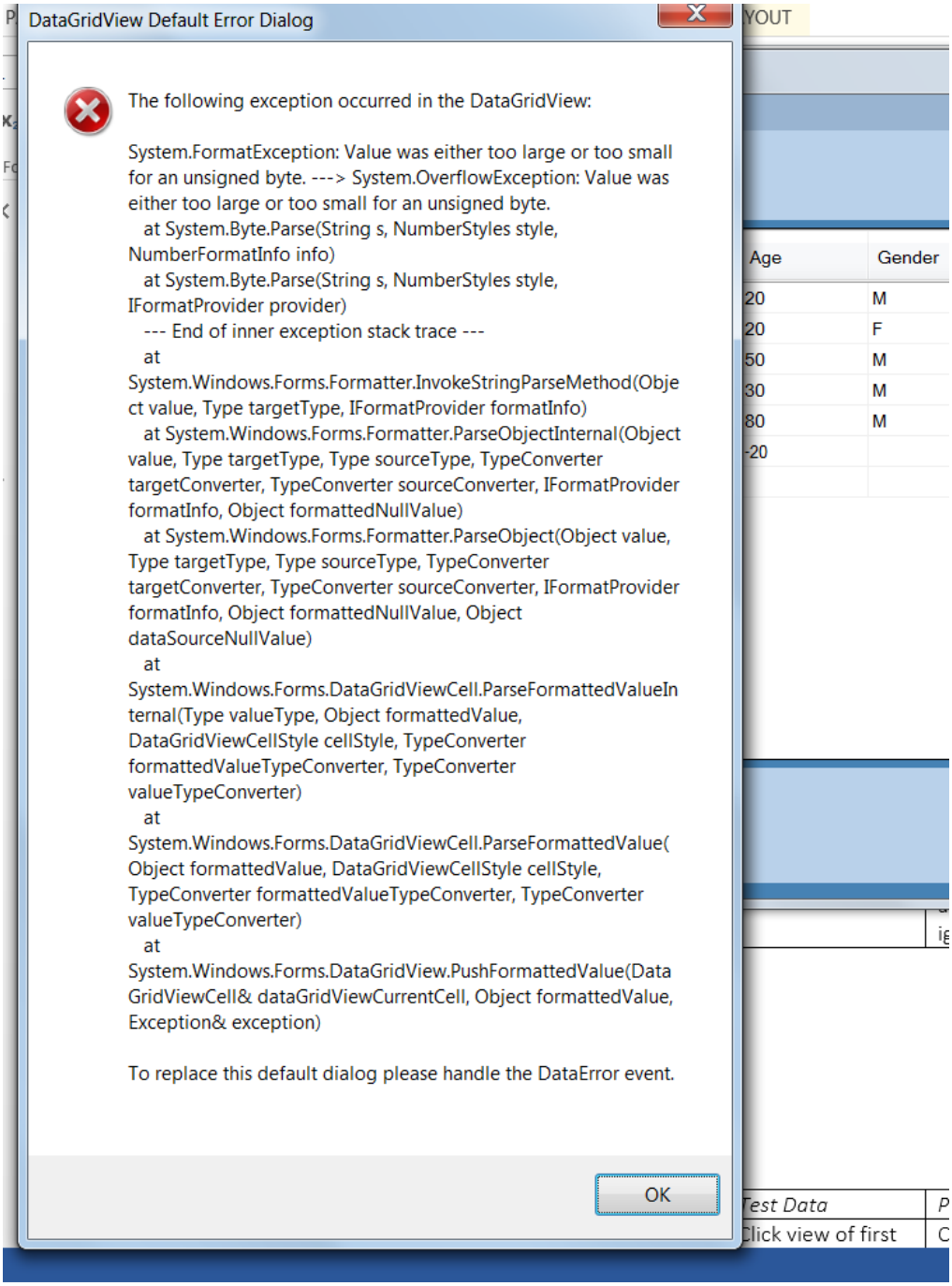
Screenshot 2



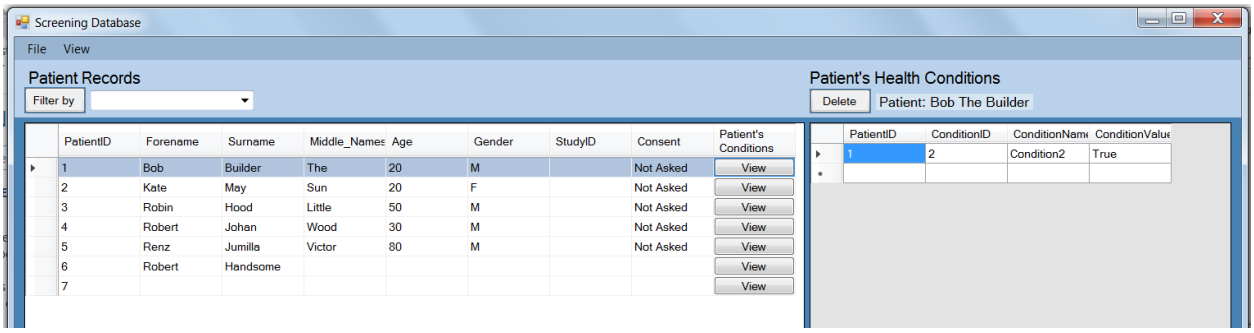
Screenshot 3



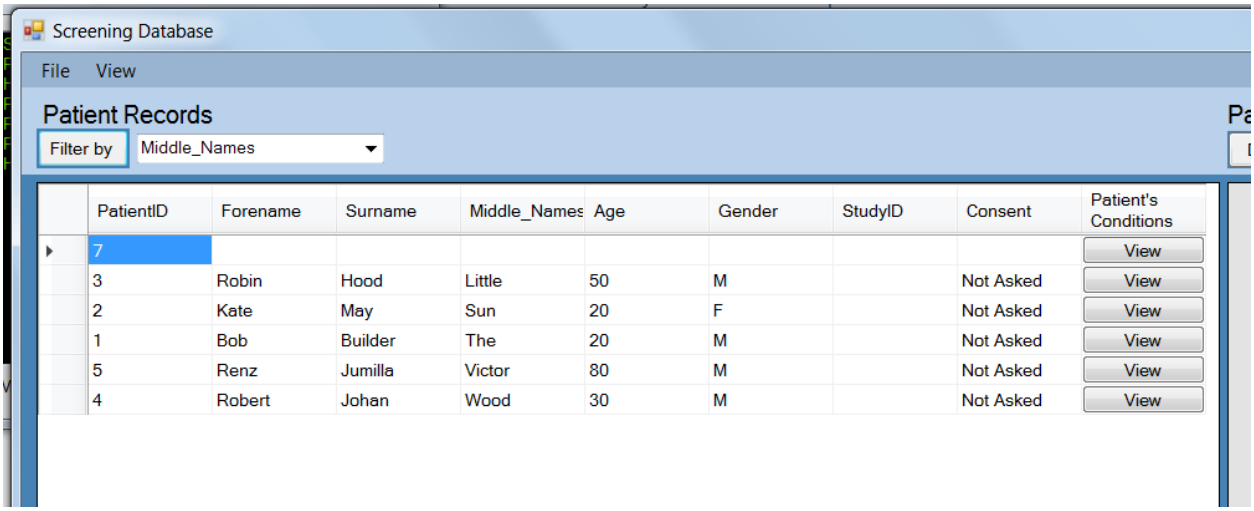
Screenshot 4



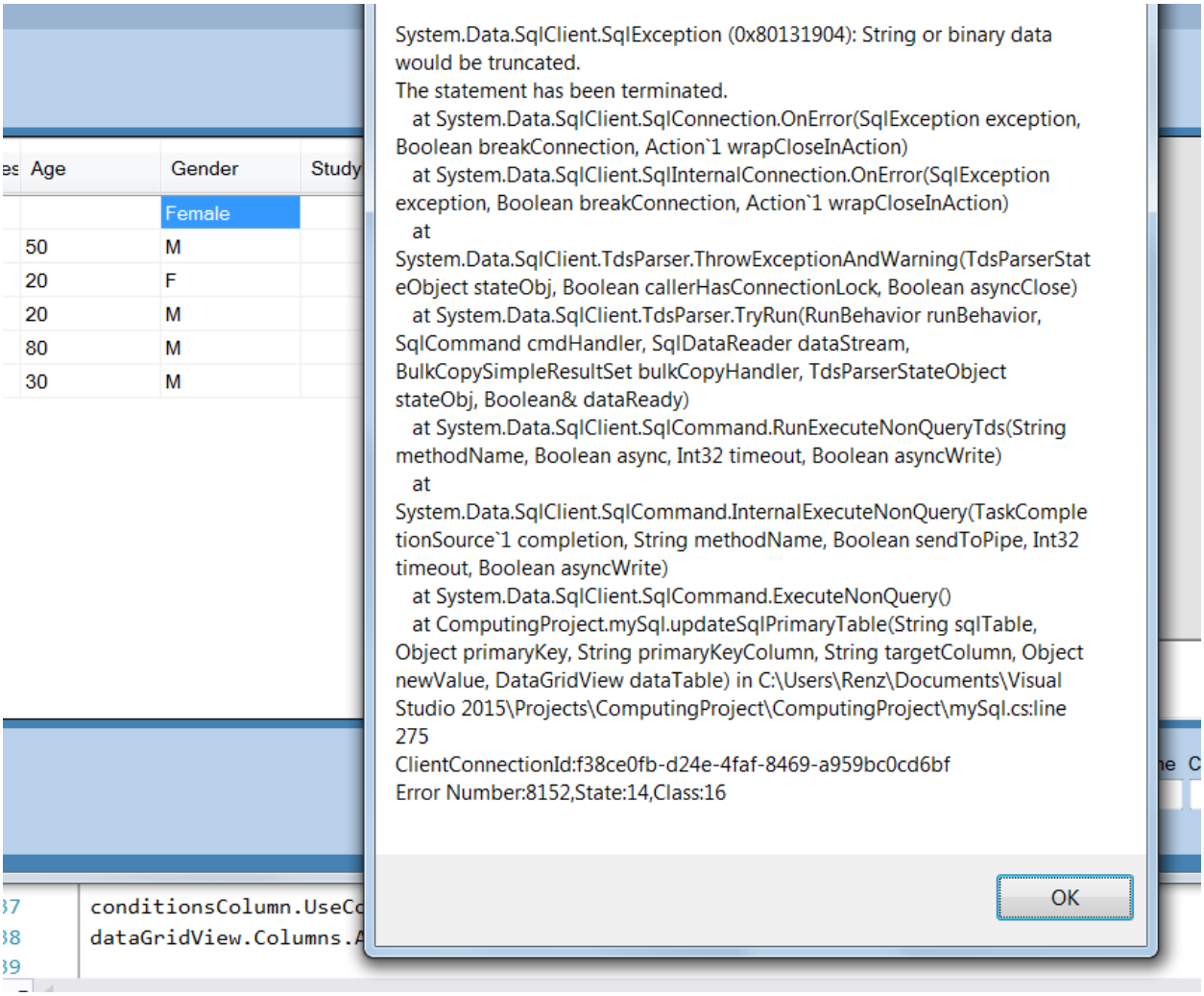
Screenshot 5



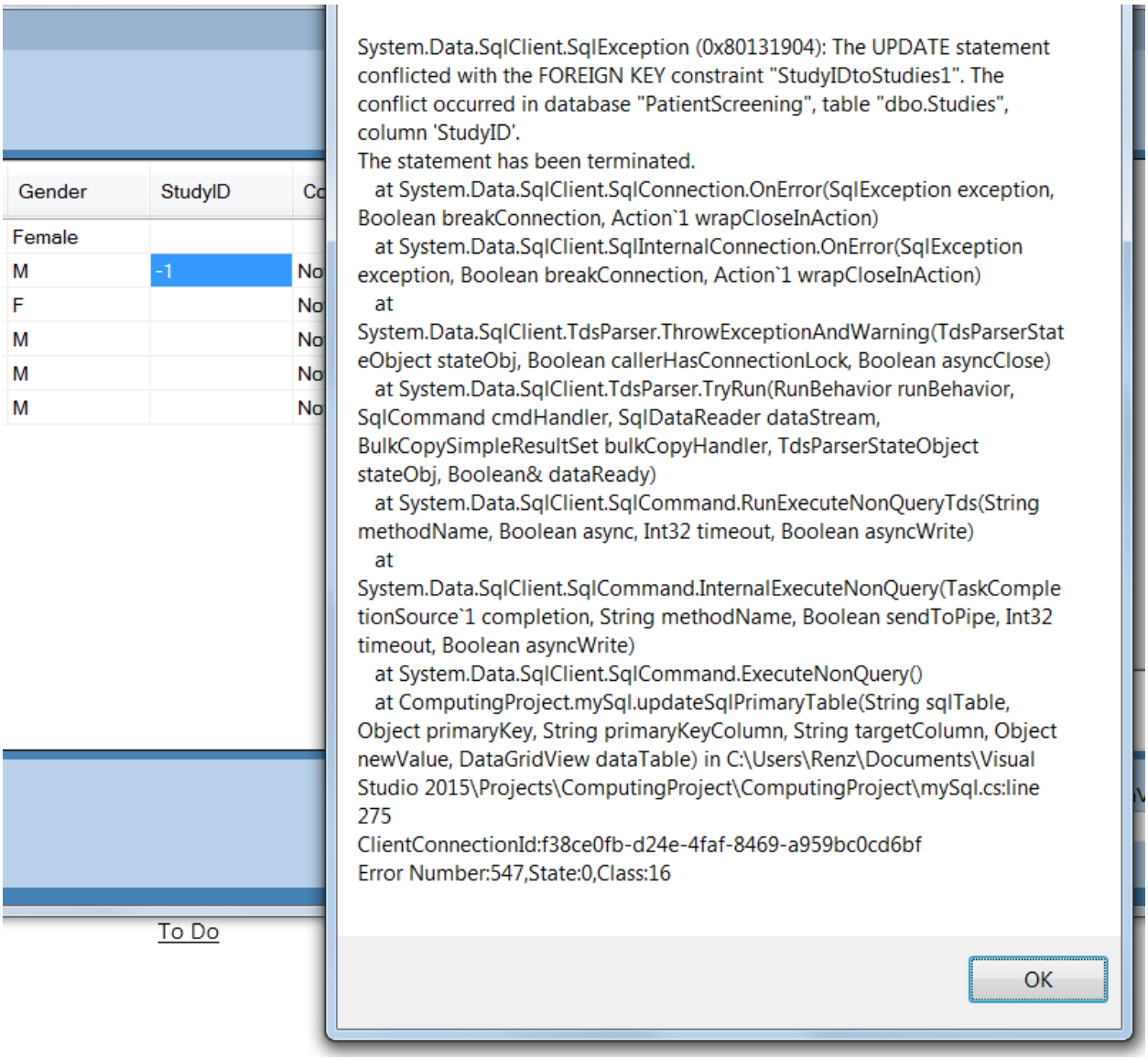
Screenshot 6



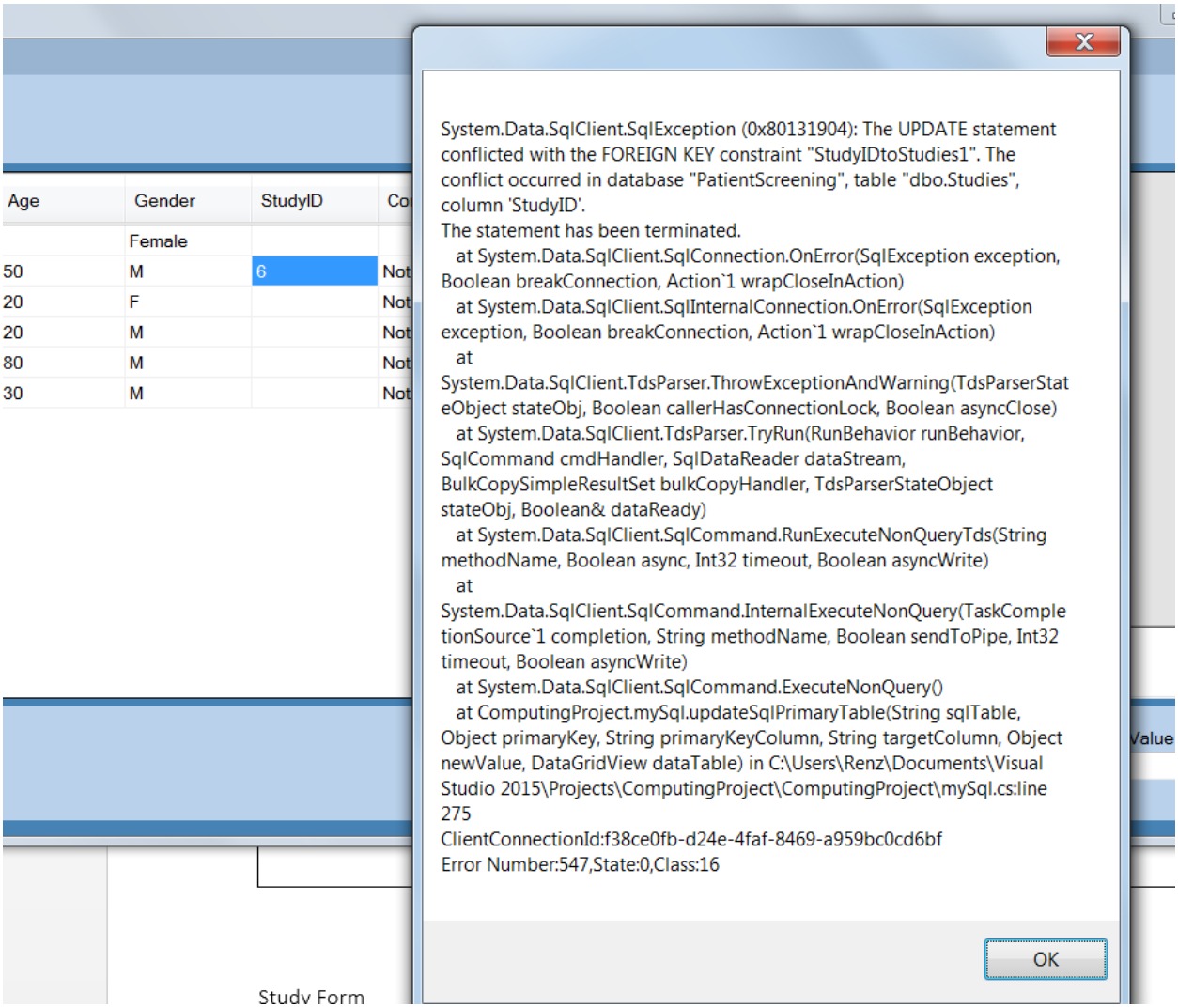
Screenshot 7



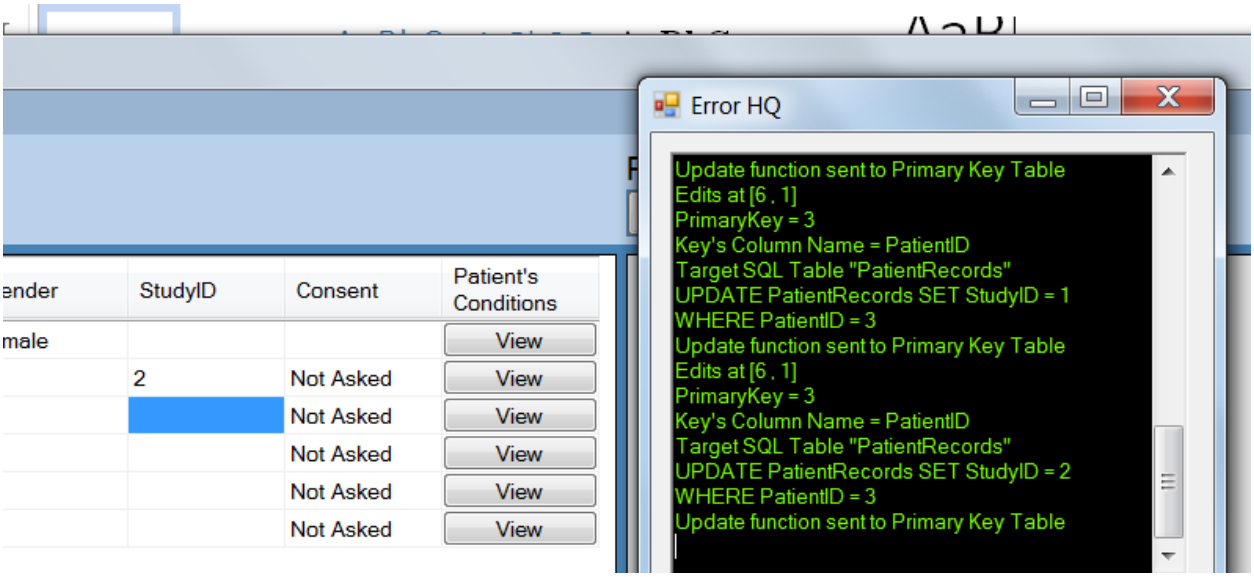
Screenshot 8



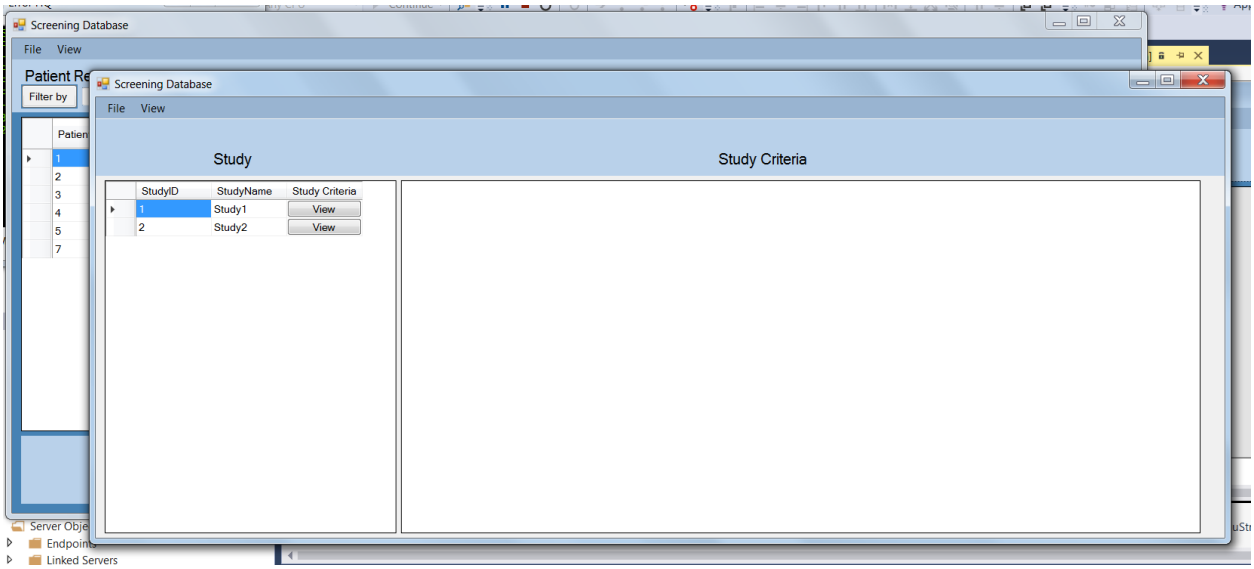
Screenshot 9



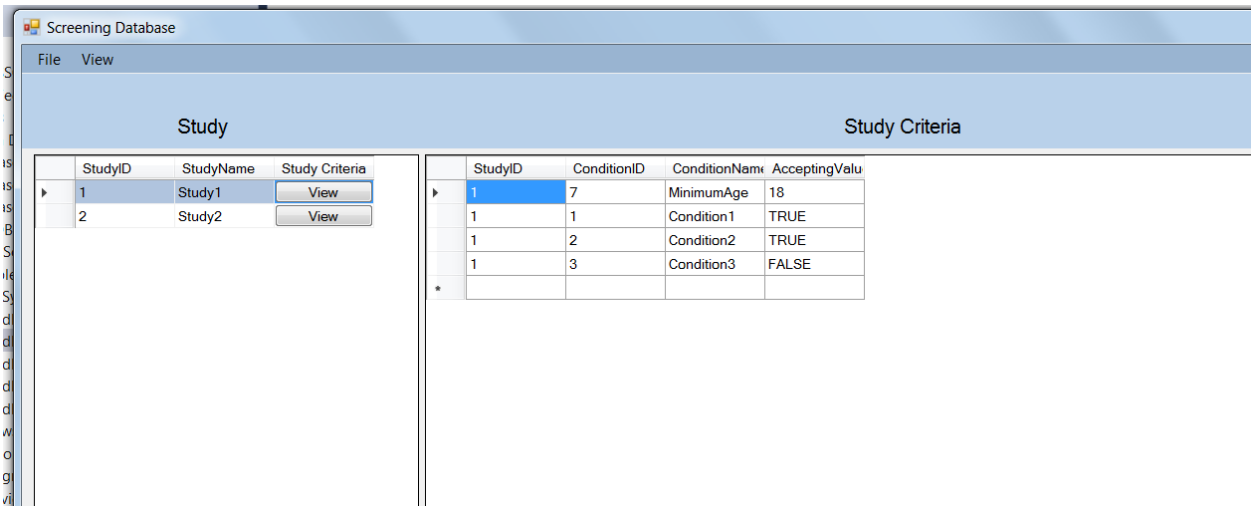
Screenshot 10



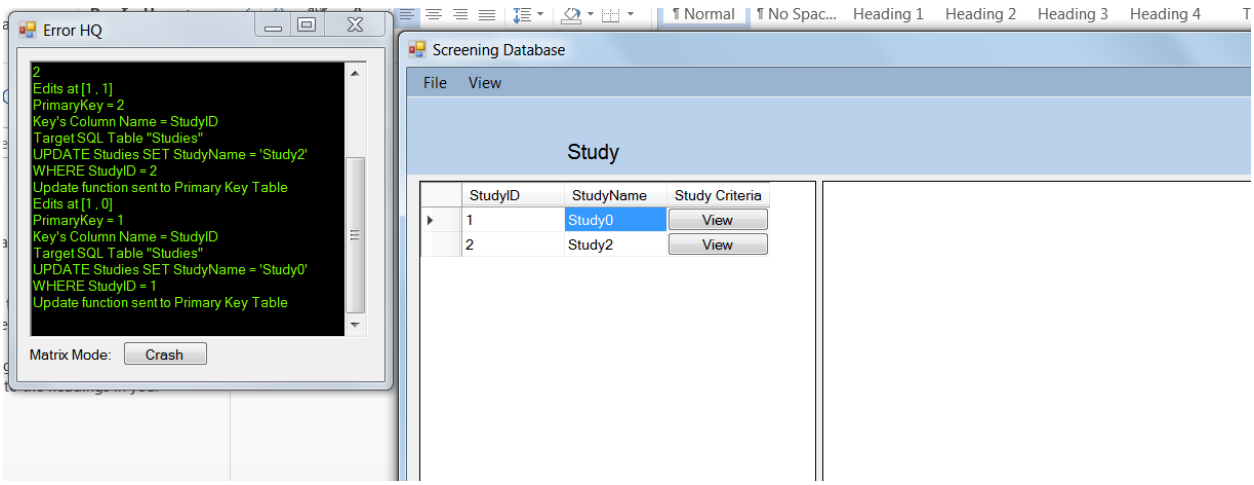
Screenshot 11



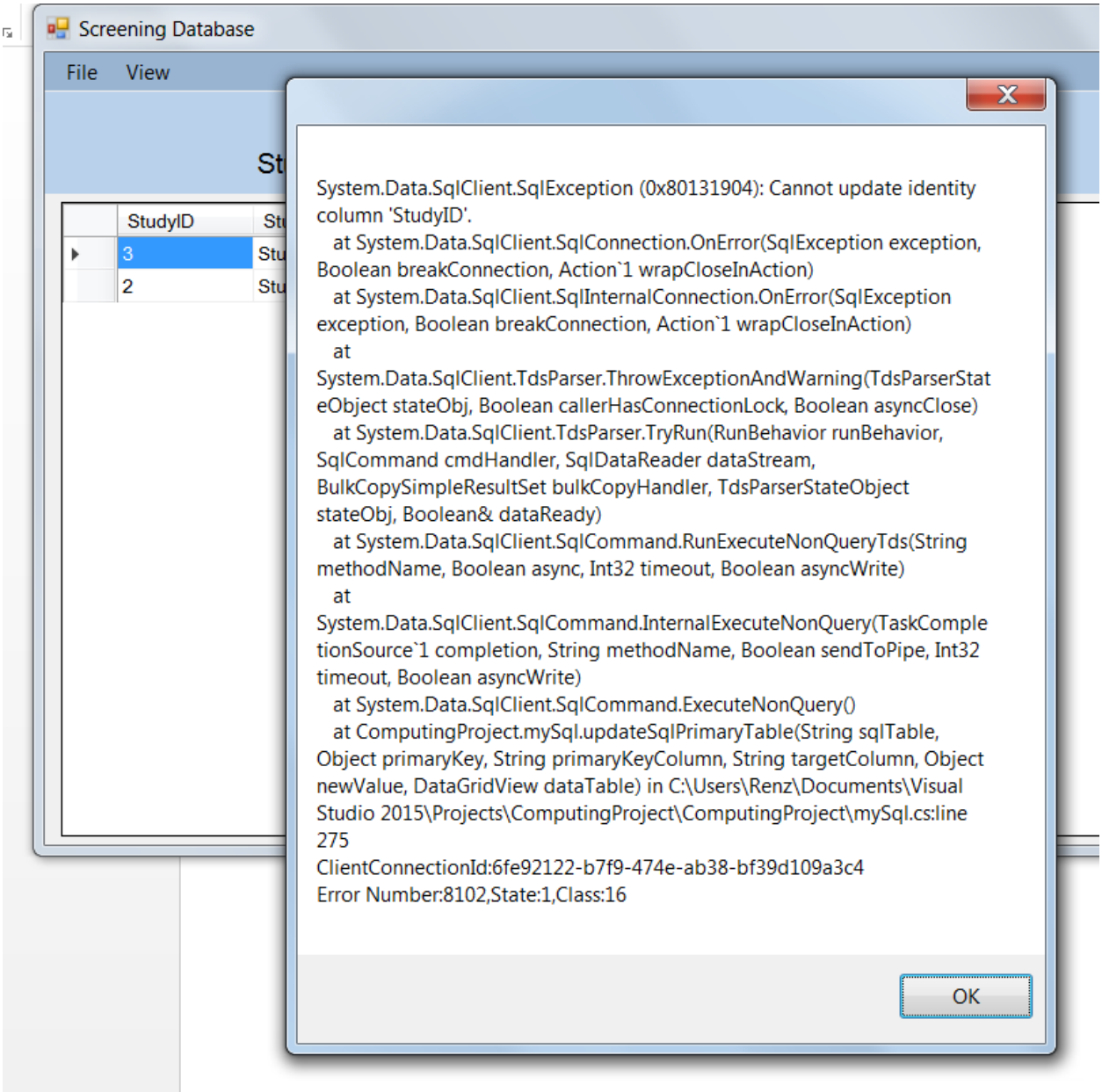
Screenshot 12



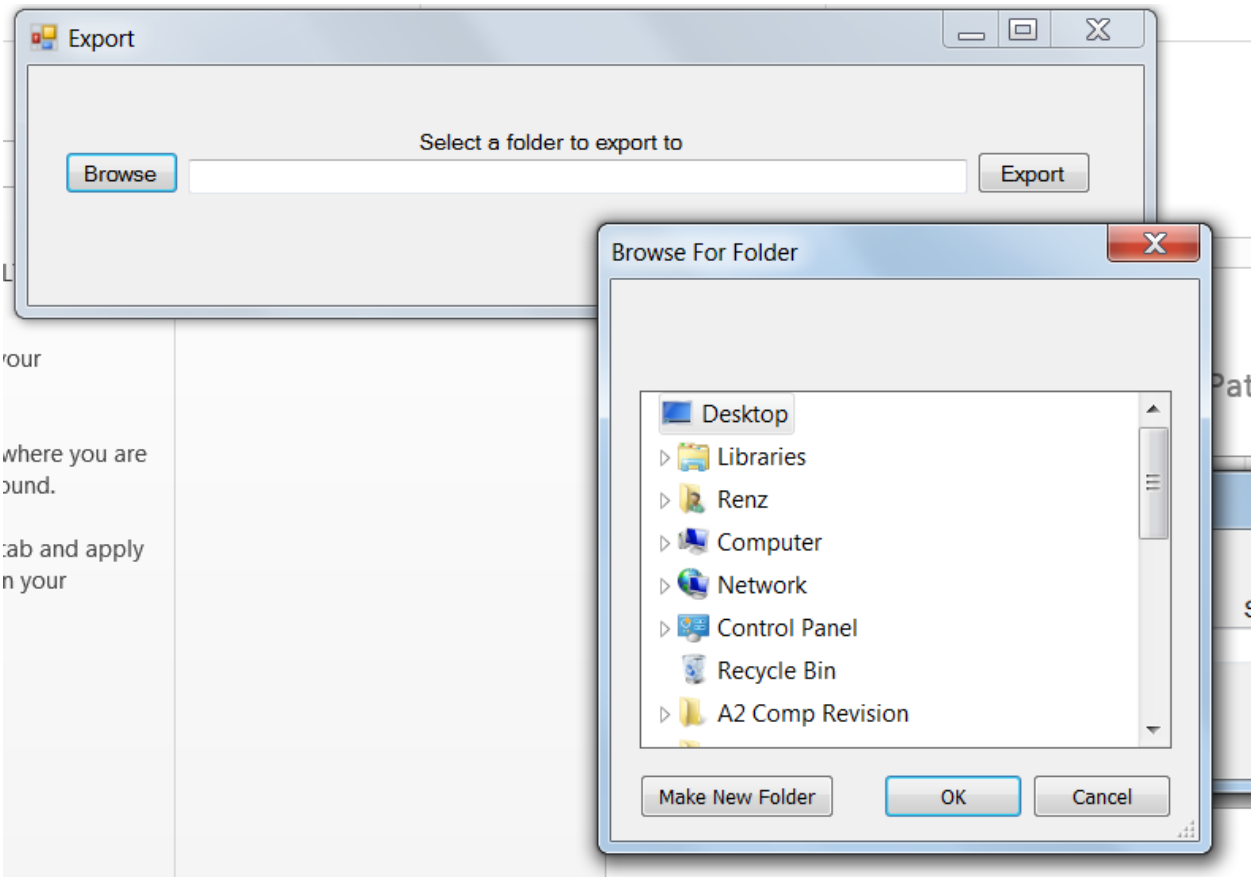
Screenshot 13



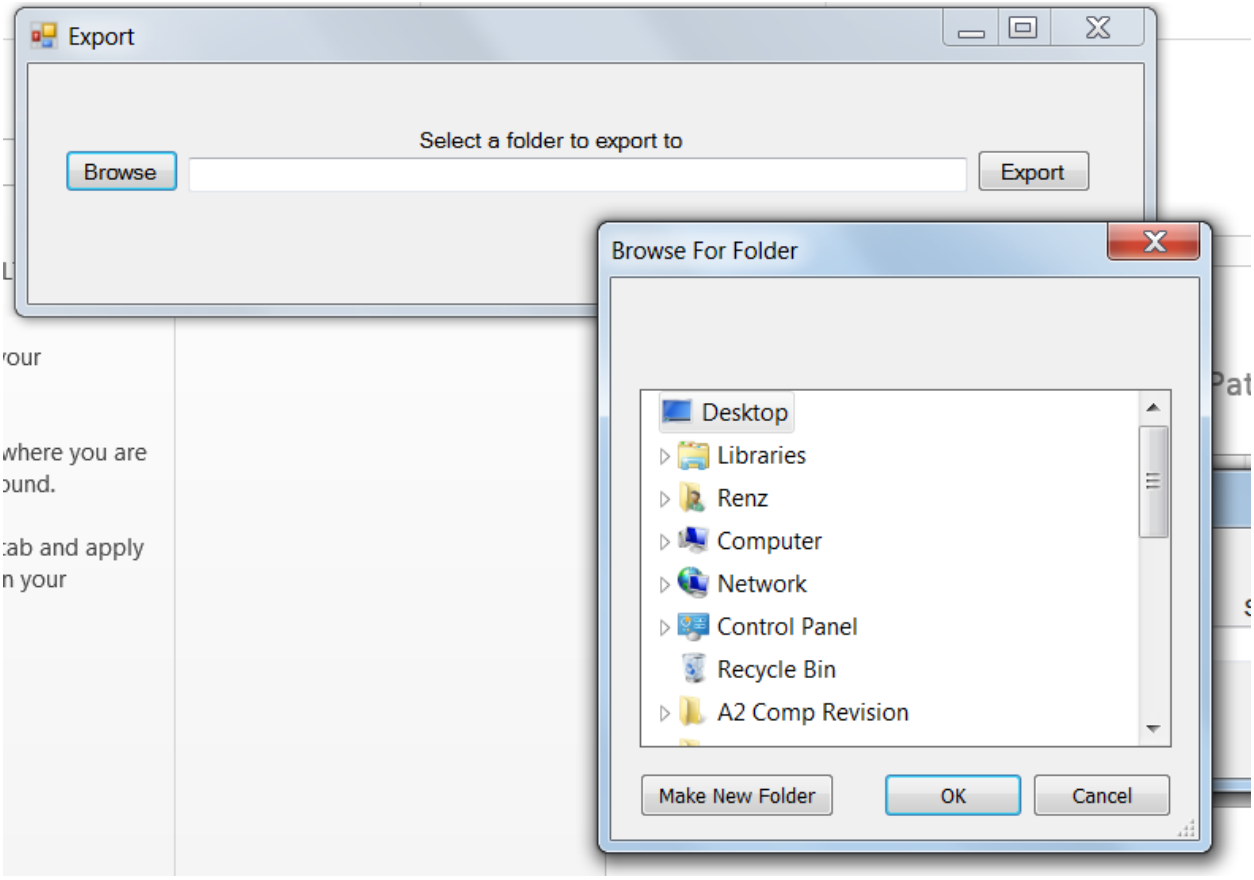
Screenshot 14



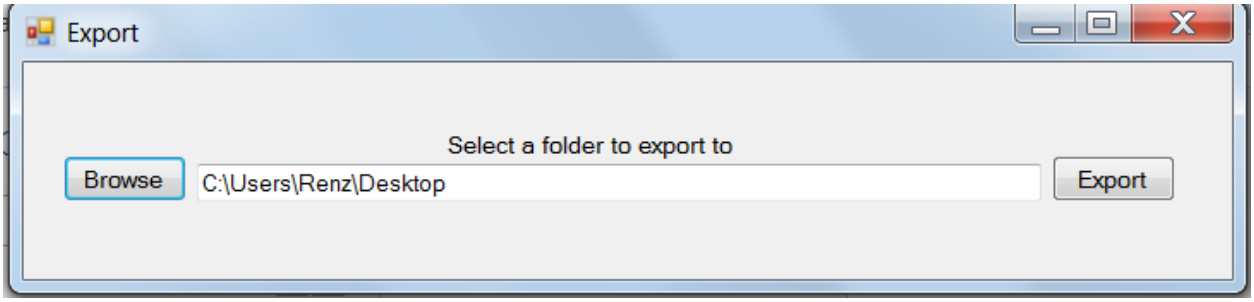
Screenshot 15



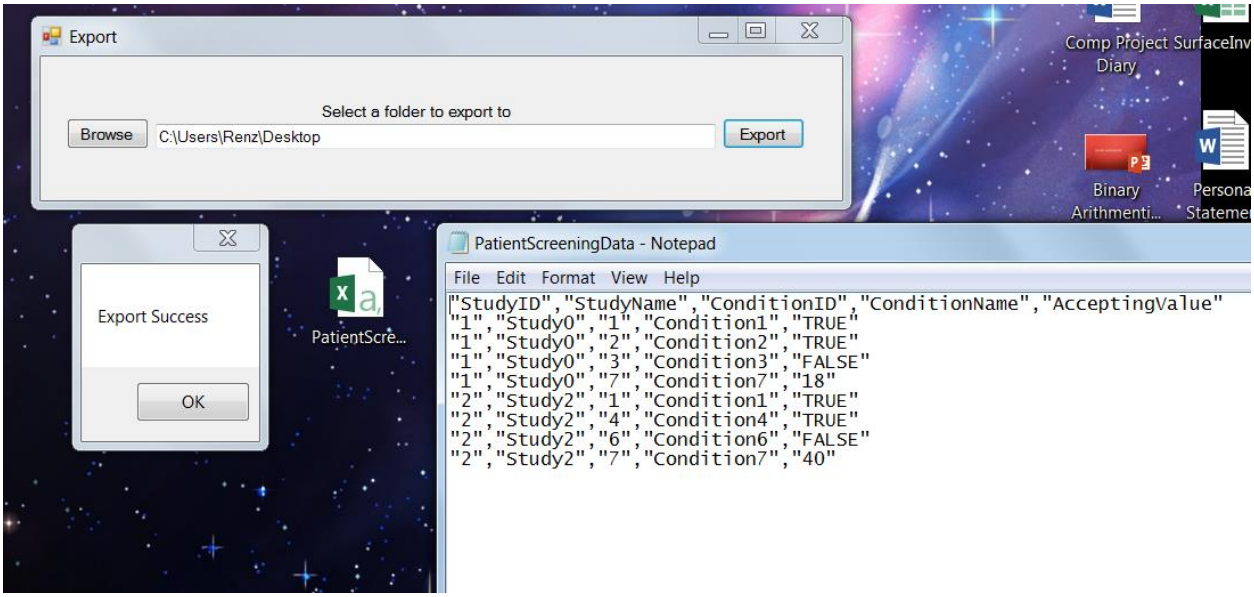
Screenshot 16



Screenshot 17



Screenshot 18

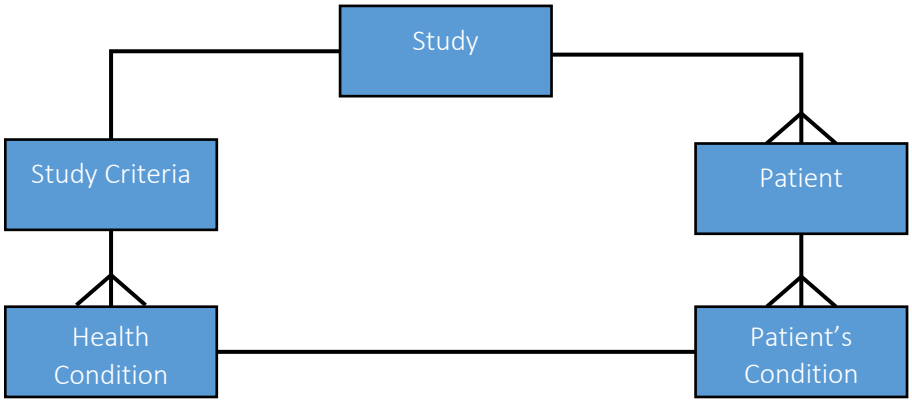


Screenshot 19

System Maintenance

The project system was designed to store patient records and Study criteria to eventually provide you with a list of recommended studies per patient.

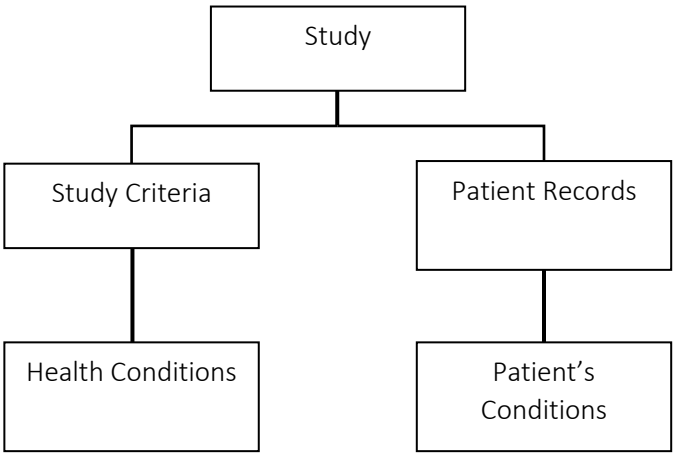
There are 3 forms in total to navigate through:



E-R Diagram 3: Relationship between entites

Description of relationships:

- A study can have many patients, and a patient can only be in one study
- A patient can have many patient conditions, and a patient's condition does not repeat for a patient
- There is one health condition for one patient condition
- A study criteria can have many health conditions, and a study criteria does not repeat a health condition
- There's one study criteria per study, and vice versa



Hierarchy 1: Relationships between entities

Hierarchy chart 1 shows how the entities are related. A Study contains a study criteria and criteria would contain a set of health conditions. A study would also contain a set of patient records, and each record would have a set of patient conditions. Patient conditions are like health conditions but belongs to patients.

Detailed Algorithm Design

The patient's form mostly consisted of event handlers. Main entry point for patient form is mainInit(). This sets out the initial state of the form by populating the table interfaces with patient records and patient health conditions.

addPatientEventHandler() adds an empty record when the Add button is pressed. This function then tells the SQL server about the new row, and stores it. Finally, the DataGridView which serves as the table interface, refreshes so that the user can see the changed table.

A patient's health conditions are displayed when the view button next to their row is clicked. The procedure first runs the patientConditions() function from myFunctions class to obtain the patient's conditions. The function does this by combining data from the PatientRecords table and PatientConditions table in the sql database. The conditions are then displayed on the right table.

After this, the recommendation algorithm is run using the conditions to see which study criteria they meet. It first runs relatedStudies() to find related studies, then checks to see if their condition values match the patient's condition values. Any related studies are then displayed under the detected studies text box.

- recommendStudy()

The process is the most significant algorithm in the project. It needs to select a suitable study for a patient by comparing their health conditions with the health conditions in various study criteria.

Only one patient's health condition a health condition in the study criteria to be eligible for that study. The pseudocode below shows how it will work.

```
//------------- Recommendation Algorithm -------------------
        studyTextBox.Text = "";
        foreach (DataRow x in conditionsDataTable.Rows)
        {
            //Test to see if patient Condition is in a criteria
            int conditionID = Convert.ToInt32(x.ItemArray[1].ToString());
            ArrayList recommendedStudies = new ArrayList();
            DataTable relatedStudies = new DataTable();
            relatedStudies = mySql.relatedStudies(conditionID);
            foreach (DataRow r in relatedStudies.Rows)
            {
                string studyConditionValue = r.ItemArray[3].ToString().ToUpper();
                string patientConditionValue = x.ItemArray[3].ToString().ToUpper();
                //If a patient condition is in a criteria, check if they have same
value
                if (patientConditionValue == studyConditionValue)
                {
                    int studyID = Convert.ToInt32(r.ItemArray[0].ToString());
                    recommendedStudies.Add(studyID);
                    myFunctions.hint("Related Study = " + studyID);
                    string studyName = mySql.getStudyName(studyID);
                    studyTextBox.Text += studyName + Environment.NewLine;//Display
study on form
                }
            }


        }
```

- exportCrtieriaToCsv()

This function converts the contents of all existing study criteria into csv format. It first gets the data from the database, and then uses data to build a list of CSV style rows. Finally once all the rows are built it's written into a CSV file.

```
static public void exportCrtieriaToCsv(string outputPath, string sqlTables)
        {
            connection.Open();

            // Connect to database and select all tables
            string selectString = "SELECT Studies.StudyID, Studies.StudyName,
HealthConditions.ConditionID, HealthConditions.ConditionName,
StudyCriteria.AcceptingValue FROM Studies, HealthConditions, StudyCriteria WHERE
Studies.StudyID = StudyCriteria.StudyID AND HealthConditions.ConditionID =
StudyCriteria.ConditionID ORDER BY StudyID, ConditionID ASC";

            SqlCommand sqlSelect = new SqlCommand(selectString, connection);
            SqlDataReader sqlDataReader = sqlSelect.ExecuteReader();

            //Create .csv file
            string filePath = outputPath + "\\" + "PatientScreeningData.csv";

            // Create CSV stream
            StreamWriter streamWriter = new StreamWriter(filePath, false,
Encoding.Unicode);//Stream writer will not allow special characters
            DataTable dataSchema = sqlDataReader.GetSchemaTable();
            string csvRow = "";
            string csvColumn = "";
            int columnCount = sqlDataReader.FieldCount;

            //------------ Write to CSV file ------------

            //Write column headers
            for (int i = 0; i < columnCount; i++)
            {
                csvColumn += "\"" + sqlDataReader.GetName(i) + "\",";
            }
            csvColumn = csvColumn.Remove(csvColumn.Length - 1);
            streamWriter.WriteLine(csvColumn);

            //Write rows
            while (sqlDataReader.Read())//Repeat for all rows
            {
                csvRow = "";
                for (int i = 0; i < columnCount; i++)//Construct row
                {
                    //csvColumn = sqlDataReader.GetName(i);
                    csvColumn = "";
                    csvColumn = sqlDataReader.GetValue(i).ToString();//Get column item
                    if (csvColumn.GetType() == typeof(String))
                    {
                        csvColumn = "\"" + csvColumn + "\"";//Add double quotes if
item is a string
                    }
                    csvRow += csvColumn + ",";//Add item to row
                }
                csvRow = csvRow.Remove(csvRow.Length - 1); //Remove comma at end of
row
                streamWriter.WriteLine(csvRow);
            }//End while

            MessageBox.Show("Export Success");
            //Close connections
            streamWriter.Close();
            connection.Close();

        }//End exportAllToCsv
```

- validatePatientRecord()

Carries out validation checks on users input into a certain column on patient record table eg. If user changed a cell under the age column, it checks if its between 0-120.

```
static public ArrayList validatePatientRecord(string sqlTable, string columnName,
object input)
        {
            ArrayList errorList = new ArrayList();
            //Check input for each attribute is correct

            switch (columnName)
            {
                case "PatientID":

                    if (input != null)
                    {
                        int key = Convert.ToInt32(input);


                        myFunctions.hint("key = " + key.ToString());
                        int duplications = mySql.duplications(sqlTable, columnName,
key);
                        if ((duplications > 1))
                        {
                            errorList.Add(columnName + ": ID should be a unique
number");
                        }

                        if (key < 0 || key > 9999999)
                        {
                            errorList.Add(columnName + ": ID should be a positive
number between 0 - 9999999");
                        }


                    }

                    break;

                case "Forename":
                case "Middle_Names":
                case "Surname":
                    if (input != null)
                    {
                        if (Convert.ToString(input).Length > 50)
                        {
                            errorList.Add(columnName + ": No. of characters should not
exceed 50");
                        }
                    }
```

```
                    break;


                case "Age":
                    if (input != null)
                    {
                        int age = Convert.ToInt32(input);

                        if (age < 0 || age > 120)
                        {
```

Appraisal

Comparison of Project Performance against Objectives

Specific Objectives

Objective	Met	Comment
Filter data in numerical or alphabetical order	Yes	Used built in functions made it easy to use
Colour scheme and layout similar to Epic system	Yes	Looks very professional and fits in well with working environment. This was the most challenging part as it meant the navigation had to be similar to excels input features
Calculate patient's most likely study within 2 seconds	Yes	Gives general idea of what a patient's study is
Ability to record and view records	Yes	Both the study form and patient form can view records
Proposed system will be able to cascade updates across other tables	No	Untested, ran out of time and was difficult to test
User will be able to delete and modify records	Yes	Does the job, but there are errors
User will be able to find a patient name within 2 seconds	No	Did not implement, ran out of time but would assume it to be an easy implementation
The system will have the option for a password	No	Ran out of time
The user will have the option to print	No	Ran out of time, and needed further research for mail merging which took time

General Objective

The overall objective was to create a database system which will allow the user to store, edit and filter patient records and study criteria, with the internal ability to automatically provide a patient's recommended study based on existing patient records and study criteria.

The technical solution failed in storing new items in the study criteria. Although the technical solution allows users to view existing studies, and change attributes permanently, they could not add a new study criteria themselves. This would mean the programmer would have to pre-install all the studies before it is given to the user.

On other hand the overall objective has met for the patient records. It allows users to add new patient conditions and patient records as well as edit. However, there are some errors which makes the user

experience clunky. For example, inputting the wrong datatype in paitentID will not be stored, but it's still displayed by the table interface and when an error popup is opened every time during an error which gets quite annoying.

Possible Extensions

If there was more time, the technical solution would fulfill the general objectives for the study form. This would mean the ability to add new study criteria.

Another extension would be to fix errors in the patient form to smooth out the user experience. This would also mean incorporating the error form into the main form itself to eliminate error popups and rejecting invalid inputs on the table interface.

The specific objectives would also have been met. This includes writing the ability for password access and mail merge printing.

With further research, time and user testing it could've been possible to re-write the program onto mobile devices, which is what the client desired. There is a good range of reliable resources and applications which makes it easier to achieve this. The most likely mobile device however would be android as it is an open platform with a lot of community resources, which would make app development quicker.

Analysis of User Feedback

The program was tested by the client, Sarah, who was pleased with the overall layout of the program. She found that adding and editing patients were easy but found the constant stream of error popups made the user experience clunky. Navigation of between forms flowed as well.