

NEA Exemplar Project

Project title:

'RED (Reverse Engineering Database)'

GCE A-level Computer Science (7517)

Introduction

This NEA exemplar project is provided to give teachers an indication of the type of project that students could complete. This A-level specification is for first assessment in June 2017.

This exemplar project should only be used to enable teachers to commence planning work for the NEA. As a result teachers should use this only as a guide to the forthcoming live assessments.

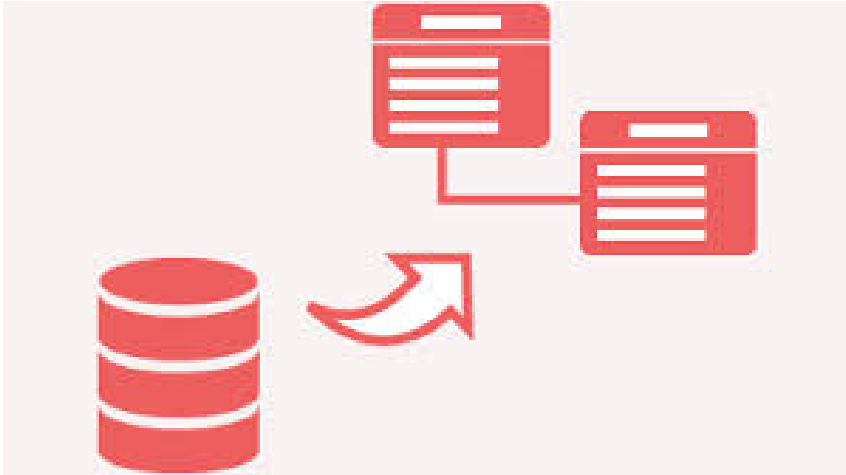
This project is provided as an example only. It cannot be used to accurately determine standards in the future.

Teacher Standardisation events will be used to prepare teachers for the first NEA assessment. At these meetings teachers will be made aware of the standard.

Note

This is an exemplar project that has been produced from a project that was entered for the previous specification. Sections from the old specification that are no longer required have been removed. The subsections left might contain elements that are not fully required by the new specification but give a good idea as to what the contents of a project might be. As this project has been adapted from the old specification and not standardised we have indicated which level it would be marked in for each section of the NEA.

Project RED – (Reverse Engineer Database)

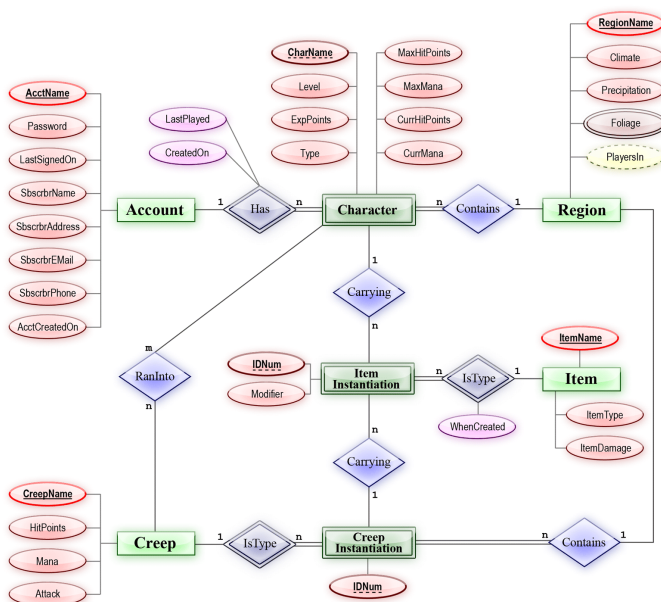


Analysis:

Introduction:

Entity-relationship models (ER) are a key part of software engineering and are used all the time in big database projects, amongst other things. They are one of the core data models and they are made up of two fundamental parts: the entities that exist in the database or system and the relationships between these entities. The entities and relationships can also have attributes, which can be anything that is useful to know about the entity or relationship.

For example, an entity could be an employee of a company; one attribute of the employee could be that his name is Henry and another could be his home address. An example of a relationship would then be that Henry *works for* Google, where Google is a different entity and the relationship is highlighted in italics. This clearly shows how the two entities Henry and Google relate to each other. Shown below is a visual example of an entity-relationship diagram.



⊥ — Exactly One

⊖ — Zero or One

⊃ — Zero, One, or More

⊣ — One or More

There is also another notation called Crow's Foot, where Crow's Foot diagrams represent entities as boxes, and relationships as lines between the boxes. Different shapes at the ends of these lines represent the type of the relationship.

Entity-relationship models can be used to map out the relationships between different tables in a database and to show how the primary keys and foreign keys match up, which is what makes them such an essential tool in software engineering. They can,

Each box in the diagram represents a different table in the database with the primary key for each one located at the top of each box. The arrows between the boxes represent the relationships between the different tables in the database; note that some boxes don't have any arrows connected to them, which means that there are some tables in the database which are independent of the others. Where it says FK1 or FK2 in the boxes, it is identifying foreign keys in the table. These are primary keys from other tables and it is through these foreign keys that you can work out the relationships in the database.

The software packages mentioned above are just the most frequently used ones and the ones that my client is using. Microsoft SQL Server can be replaced with MySQL, which is a free alternative, but it does not have the same amount of features as Microsoft SQL Server, for example it does not have the same smoothness with Windows and other Microsoft software that SQL Server has. As for an alternative for Microsoft Visio, there are lots of ER tools that are available online, such as Lucid Chart³, but these programs do not let you connect a database to them and then draw a diagram of your database; they are designed for people who are creating new databases who want to draw the ER diagram first. However, there are others such as SchemaCrawler⁴, which can draw diagrams of existing databases for you.

Identification of End Users:

The primary end user of this project is my father, Adrian; he is an IT consultant and often works with databases. The software that he uses is mentioned above in the Description of Current System section. After speaking with Adrian I have found out that he often encounters problems with Microsoft Visio and that it often creates incorrect diagrams of his database. He has also demonstrated this problem to me on his computer and shown me that when he attempts to create an entity-relationship diagram, some of the relationships and foreign keys are missing. He has also expressed his concern to me about the cost of Microsoft Visio and how the immense cost of it will mean that lots of people will be unable to afford it. He has therefore asked me to devise a solution.

As well as Adrian, my project has other end users. The problem that I am solving is one that affects lots of other database engineers who want to make an ER diagram. An ER diagram is necessary to see how data flows through a database and how it is stored, so they will also be able to benefit from my project, especially from Solution 1, which provides them with a free ER drawing tool that can connect to their database.

³ <https://www.lucidchart.com/documents/edit/2c51793e-bc22-4201-8897-2a6622be6fe5#?demo=on>

⁴ <http://schemacrawler.sourceforge.net/diagramming.html>

User Needs:

- A software tool that is preferably free to use, that allows them to connect their database to it and then produces an ER diagram of the database.
- The software must be available online and be easy to download and run.
- Be able to save the diagram to their computer or keep track of the diagrams the user has created, so they can view the diagrams they have created in the past.
- The ER diagram must have the same standards as other ER diagrams, so that the users can understand it and can compare it to diagrams created with different programs.
- Some customisation of the diagram is allowed, e.g. changing colours or type of ER diagram (Crow's Foot or normal).

Acceptable Limitations:

- Database selected for reverse engineering must contain full 3NF relationships - i.e. for SQL Server both Primary and Foreign Keys must be defined and the database must be normalised to the 3rd level (for Solution 1).
- Program might not be able to connect to all databases.
- User's DB must also be able to be accessed from outside the LAN the DB is hosted on.
- User must have sufficient privileges to access whichever database they want to connect to.
- Customisation of diagram will be limited due to complexity of programming.

Data Sources and Destinations:

The main data source for my program is the user's database, from which my program will extract the data that it needs to draw an ER diagram. The extracted data will then be stored locally in the program in a C# data structure called a datatable, which is just like a normal table in a database, but is stored within the program in RAM rather than on a server. Storing the data this way also allows me to easily iterate through the datatable's rows without having to do another SQL Connection in my program. It is also much faster than a DB table.

The data that is coming out of my program is in the form of a JPEG image of the ER diagram that the user will draw. They will also be able to save it for review later.

Data Volumes:

My program will be storing data locally in the program during runtime in the form of two datatables, which will be filled with the results of the queries, and it will also be storing a JPEG image of the diagram to a file location specified by the user. Both data volumes will vary depending on the size of the database that the user has chosen to connect to.

For example, shown below is an example of some results that the tables query would return. This query creates a list of all the tables in a given database, as well as some other information about

each table. Each field stores up to 20 characters apart from the description field which stores up to 50. So if a database has exactly 100 tables, the size of the results from the tables query stored in a local datatable in my program will be around 11,000 bytes.

Analysis Data Dictionary:

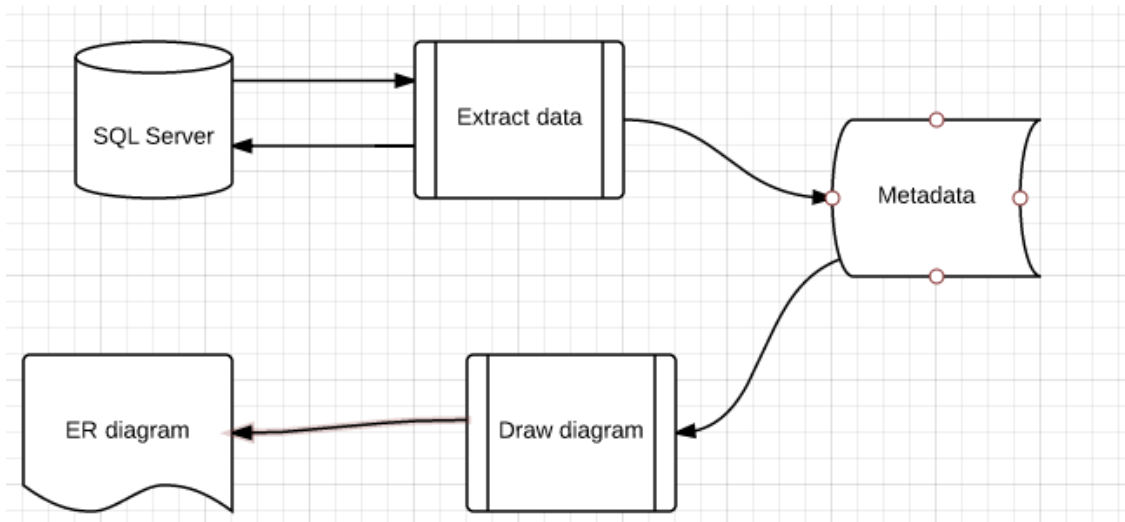
Tables:

TableCatalog	TableSchema	TableName	TableDescription
AdventureWorks	HumanResources	Employee	Employee information
AdventureWorks	HumanResources	Department	Department information
AdventureWorks	Person	Address	Addresses for customers, employees etc.

Foreign Keys:

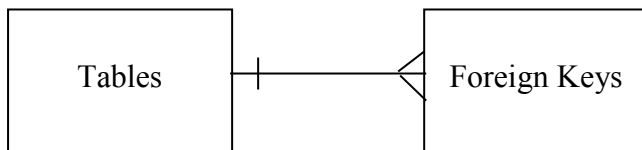
FK_Name	Schema_name	Table	Column	Reference d_schema	Reference d_table	Reference d_column
FK_Employee_Contact_ContactID	HumanResources	Employee	ContactID	Person	Contact	ContactID
FK_ProductInventory_Product_ProductID	Production	ProductInventory	ProductID	Production	Product	ProductID

Data Flow Diagram:



ER Diagram:

My project will be querying the user's database and then creating two tables of its own to store the extracted data.



Object Analysis Diagram:

As I will not be creating any of my own classes for this project, an object analysis diagram is not relevant.

Objectives:

1. Allow the user to enter configurations to connect to a remote database of their choice these will include server name, database, username and password
2. Output an error message if the program fails to connect to the database, alerting the user that they might have entered the wrong details.
3. Use SQL queries to extract the metadata and keys from all the different tables and then store that data locally in a C# program written in Visual Studio.
4. Allow the user to choose which tables they want to include in the diagram.
5. Use the data from the database to draw a unique ER diagram of the database.
6. Give the user the option to save the diagram as an image.
7. Refine ER diagram drawing algorithm, so that the diagram is clearer and easier to read (i.e. there are not so many lines crossing over each other).

Proposed Solution:

Create a custom program that is able to connect to a database, extract the metadata and foreign keys using SQL queries and then read that data into a C# program that would create an ER diagram based on the extracted data. This program would be open source and would be available for free on the internet. The user would not have to download any other software to use it, but would have to be using a Windows operating system. There would be some limitations as to the type of database the program would be able to connect to, as specified in the acceptable limitations.

Evidence of Analysis:

So that I could properly understand the problem that my end user was facing, I decided to interview him about it. During the interview he showed me the basics of Microsoft SQL Server and also some examples of the problem in Microsoft Visio, where the ER diagrams are not being drawn correctly. He also informed me that the latest update of Microsoft Visio does not allow you to create ER diagrams at all, meaning that my project is ever more important. The interview is shown below:

What are you having problems with at the moment?

One aspect of my job as an IT Consultant is that, as we implement a new system for the customer, we also need to move their data from their old system into the new one. This is known as Data Migration and involves being able to understand how the data is stored in both the old and new systems. It is almost always the case that the data will be structured differently and the best way to review these before/after data structures is through the use of a user-friendly diagram. Entity Relationship diagrams are one way of presenting this information in a manner that facilitates the discussion with the business users about how they work with the data day to day in their old system and how they will in the new system.

How do these problems affect your work?

Personally, I have a copy of Microsoft Visio, which has a reverse engineering facility, but unfortunately a lot of my customers do not and it is often not possible for me to connect my laptop to their corporate network. Additionally, they will not permit me to take a copy of the database to restore onto my laptop.

Is there anything else that you don't like about the current software that you use?

Visio is also expensive to buy and I have quite an old version (2003) which can be buggy at times and miss relationships. This is especially true when trying to connect to more recent versions of SQL Server, like 2008 and 2012.

Would a custom program that lets you draw ER diagrams of your database solve the problems that you are currently having?

A free-to-use web-based utility that performed this reverse engineering and produced an Entity Relationship diagram would help me in these circumstances. In turn, the use of the ER diagrams enables better understanding of existing business processes and data. Poor quality data is often

cited as a principal reason for new systems' failure, so a tool that helps me understand data structures within the DB and that is portable from customer to customer would be very beneficial.

Is there anything specific that you would like this program to do?

Ideally the program would have the following features:

- *Be able to connect to many different databases by allowing the user to select server name, username etc*
- *Be able to extract details of the tables, columns and relationships from the target DB*
- *Let me select the tables to be drawn in the ER diagram - for example, I may only be interested in certain tables or schemas when migrating data*
- *Be able to draw an ER diagram representing the metadata information retrieved from the database*
- *Let me save the diagram for future reference or for distributing to other team members.*

Based on the results of the interview with my end user, it looks like I was correct to go with the proposed solution, which is to write a custom program as that is also what the end user would like. They have also explicitly said that they would like it to have certain features, such as being able to choose which tables in the database are included in the diagram, so I will need to make sure that I include that and the other features in my program. They have also requested that it will be a free-to-use application, available for download on the internet, so I will have to find a way to do this – either by creating my own small website with the file available for download or having it hosted on a site like Github, where it can also be downloaded.

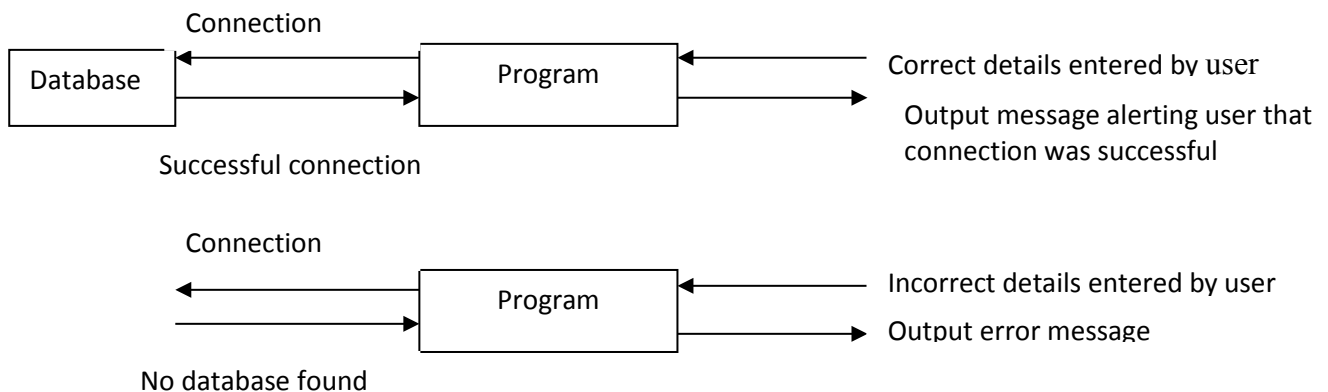
Documented Design

Overall System Design:

In order to make my project easier and so that the code works properly, I have split it into five different stages, so that I can work on just one stage at a time rather than trying to do all the different things at once. That way once I have completed programming one stage and tested that it works properly, I can move onto the next stage, not having to worry whether the previous stage will still work or not.

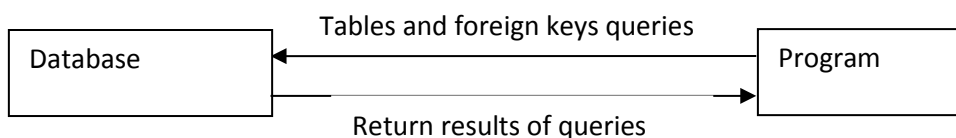
Stage 1:

The first stage of my program is to prompt the user for the details of the database that they want to connect to and then to attempt to connect to the database using the details that they provided. If their details are incorrect then the program will output an error message.



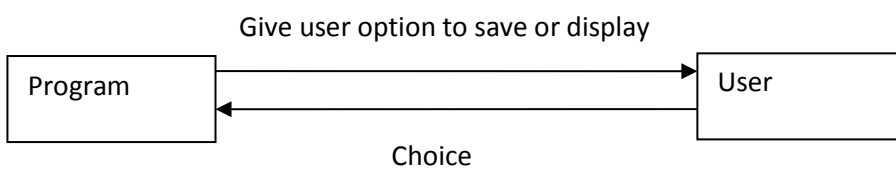
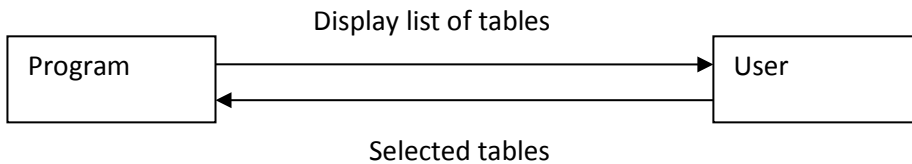
Stage 2:

The second stage is using the connection that the previous stage established to run two SQL queries on the database and then to store the results of those queries locally in the program in a C# data structure called a datatable, which is just like a table in a database with rows and columns. I will use two datatables, one for the results of the tables query and another for the results of the foreign keys query. The two queries can be seen below in the SQL Queries section.

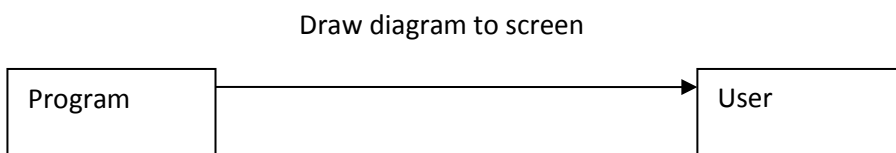


Stage 3:

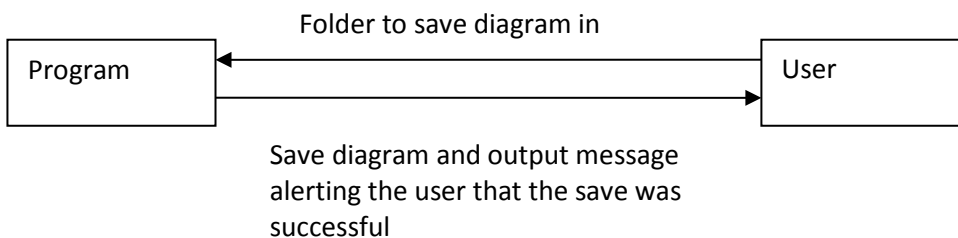
In this stage the program will show the user a list of every table in the database that they chose to connect to and give them the option of selecting which tables they want to include in their diagram. It will also let them choose to select all of the tables. A prompt will then ask the user whether they would like to save their diagram or just view it.

**Stage 4:**

Depending on what the user chose in the previous stage, the program will either draw the diagram to the screen or draw the diagram as a bitmap. The size of the diagram will depend on the number of tables that the user decides to include in stage 3.

**Stage 5:**

If the user chooses to save the diagram in stage 3, then they will be asked what location they want to save their diagram. After clicking the save button their diagram will be saved as a JPEG file to the specified folder.



IPSO Chart:

Inputs	Processes	Storage	Outputs
Database details to connect to	Connect to database server	Temporarily store metadata obtained from SQL queries in local datatables in the program	Show user list of tables in their database, so they can choose ones to be included in diagram
Metadata from database	Extract data	Save created ER diagram to location specified by user	Display the created ER diagram to the screen
	Draw diagram		

Modular Design:

- **First Form** (Main Menu)
 - Connect and extract
 - **Second Form** (Choose Tables)
 - Confirm (use selected tables)
 - Choose save
 - **Third Form** (Save Form)
 - Enter filepath and save
 - Main menu (go back to first form)
 - Exit program
 - Choose to display
 - **Fourth Form** (Drawing Form)
 - Select all (use all tables)
 - Choose save
 - **Third Form** (Save Form)
 - Enter filepath and save
 - Main menu (go back to first form)
 - Exit program
 - Choose to display
 - **Fourth Form** (Drawing Form)
 - Main menu (go back to first form)
 - Exit program
 - Exit program

Definition of Data Requirements / Description of Record Structure(s):

My program will be storing data locally in datatables in RAM during runtime. Datatables are just like normal tables in a database, but they are quicker to work with as I do not have to make an SQL connection to be able to access them, as they are not hosted on a database. This means that it is

easy for the program to loop through the rows in the datatables and it is able to quickly access the data that is extracted from the database.

My program will also be storing data in several lists whilst it is running. The most important list will be called TableList and this list will contain all of the tables in the database. When the user chooses which tables will be included in the diagram, a new TableList will be made to replace this list. If the user chooses to use all the tables then the list will remain the same. Other important lists are ForeignKeysReferencedTable, ForeignKeysTable, relationships. These are all created and used in the getRelationships function and they will store the names of the tables which contain foreign keys and the names of the tables that have primary keys used as foreign keys in other tables, so that the relationship can be mapped between the tables.

The column headings of the three datatables that will be used during runtime are as follows:

Tables datatable:

TableCatalog	TableSchema	TableName	TableDescription

ForeignKeys datatable:

FK_Name	Schema_name	Table	Column	Reference d_schema	Reference d_table	Reference d_column

TablePositions Datatable:

table_name	x1	y1	x2	y2

Data Validation:

The data that the user will be entering is the details of the database that they want to connect to. If their data is valid and they have entered correct value into all of the fields, then the program will successfully connect to a database. Otherwise if the data that they entered is not valid, the program will fail to connect to a database and will output an error message with information on why it failed.

The user will also be entering data when they are choosing what folder they want to save their diagram in, if they enter in the wrong file path, then the save will not succeed and the program will output an error message to the user.

SQL Queries:

Extract Tables:

Returns a list of all the tables in a database, as well as some information about each table such as the description and the table schema. The query gets the data from the information tables. These tables are automatically created when you make a database in SQL Server and are updated as you add more tables and columns. They contain details about all of the tables in your database.

```
SELECT TableCatalog = tbl.table_catalog, TableSchema = tbl.table_schema, TableName =  
tbl.table_name, TableDescription = prop.value FROM information_schema.tables tbl LEFT JOIN  
sys.extended_properties prop ON prop.major_id = object_id(tbl.table_schema + '.' + tbl.table_name)  
AND prop.minor_id = 0 AND prop.name = 'MS_Description' ORDER BY TableName ASC
```

Extract Foreign Keys:

Returns a list of all the foreign keys and what a table they are located in, as well as what table and column they are referencing. This query also uses some automatically generated tables to retrieve a list of all the foreign keys. It however requires that the foreign keys have been properly defined in the database.

```
SELECT obj.name AS FK_NAME, sch.name AS [schema_name], tab1.name AS [table], col1.name AS  
[column], sch2.name AS [referenced_schema], tab2.name AS [referenced_table], col2.name AS  
[referenced_column] FROM sys.foreign_key_columns fkc INNER JOIN sys.objects obj ON  
obj.object_id = fkc.constraint_object_id INNER JOIN sys.tables tab1 ON tab1.object_id =  
fkc.parent_object_id INNER JOIN sys.schemas sch ON tab1.schema_id = sch.schema_id INNER JOIN  
sys.columns col1 ON col1.column_id = parent_column_id AND col1.object_id = tab1.object_id INNER  
JOIN sys.tables tab2 ON tab2.object_id = fkc.referenced_object_id INNER JOIN sys.schemas sch2 ON  
tab2.schema_id = sch2.schema_id INNER JOIN sys.columns col2 ON col2.column_id =  
referenced_column_id AND col2.object_id = tab2.object_id ORDER BY tab1.name ASC
```

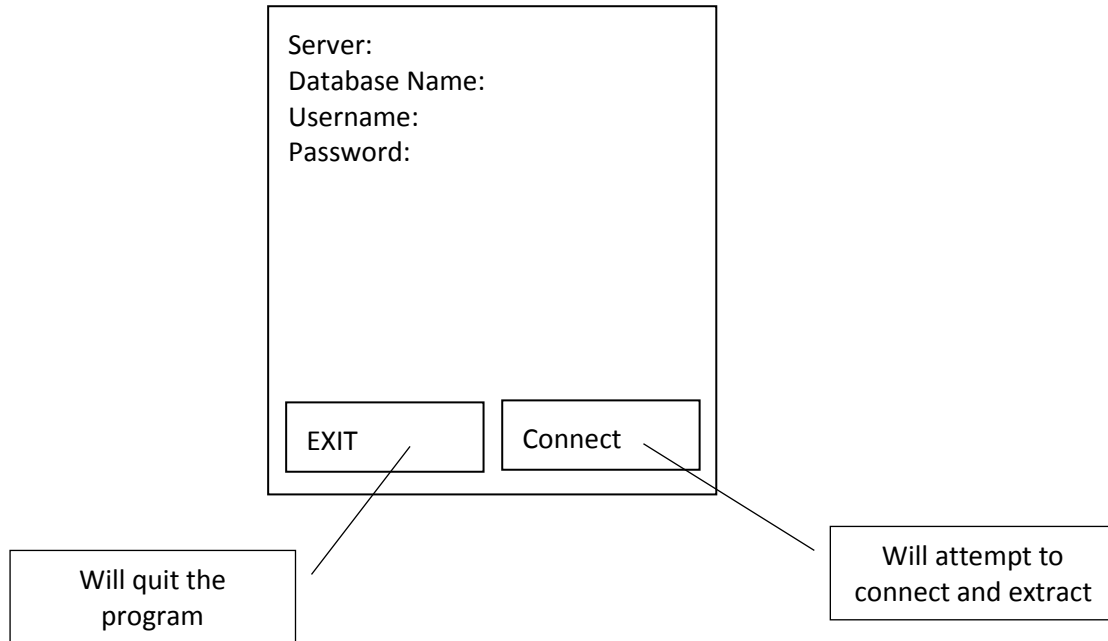
[**NOTE:** these queries have been adapted from advice given on Stack Overflow and tested against an SQL server]

Algorithms:

Extract Data:

This part of the system will extract information about tables, columns and foreign keys from the SQL database entered by the user, and will store the extracted data into local datatables for later use. It will also make a list of all the tables and foreign keys in the database.

Sketch of form:



Process:

- Firstly a connection string is made using the information that the user enters into the first form.
- Next an SQL connection is created using the connection string and then the program will attempt to open the connection, catching any errors if they do occur.
- If it's successful then it will say "connection successful", otherwise it will print an error message with information on why it failed.
- An SQL command is then declared, called `extractTables`, which uses the SQL connection the program just made and the tables query.
- An SQL data adapter is created, using the `extractTables` command, which allows for the results of the tables query to be filled into a local datatable.
- The query is then executed on the database, with any errors being caught.
- After the data is extracted a list of all the table names in the database is created by looping through the datatable that has just been filled.
- The program will then do the same thing again, but with the foreign keys query instead.

Pseudo Code:

```

Gather strings from entry form
TRY
    connect to SQL database using provided details ( new SqlConnection )
    Message 'successful'
CATCH
    Message 'unsuccessful'
extractTables (run sql statement see earlier section)
for each row in returned result
    add to tableList the table name
extractForeignKeys (run sql statement see earlier section)
for each row in returned result
    add to foreignKeyList

```

Find Relationships:

This part of the system will find all the primary key to foreign key relationships in the database and returns them in an array.

The algorithm will be written as a function, so that it can be easily called multiple times. The function takes a datatable as a parameter.

Process:

- Three lists are initialised (ForeignKeysReferencedTable, ForeignKeysTable, relationships), which are filled in later in the function.
- The program then loops through all the rows in the datatable passed into the function and adds data from each row into ForeignKeysReferencedTable and ForeignKeysTable.
- It adds the table in which the foreign key is located into one list and the table to foreign key is referencing into another list.
- Next a count variable is initialised and then the program iterates through the values in both lists adding them to another list, but it does it in a way that the corresponding tables and referenced tables are located next to each other in the list, so the relationships can still be easily read. It does this as it is much harder to return two lists than one.

Pseudo Code:

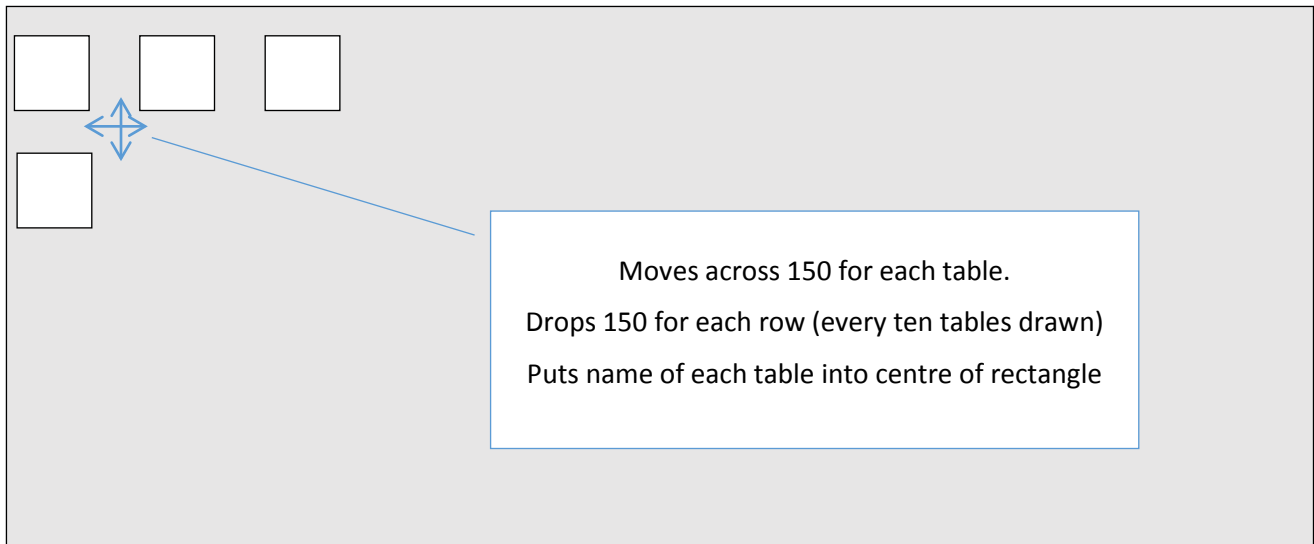
```

Create three new lists (ForeignKeysReferencedTable, ForeignKeysTable and relationships)
For each row in foreignKeys datatable
    Add 'referenced_table' to ForeignKeysReferencedTable list (adds data in 'referenced_table' field in foreignKeys
datatable to list)
    Add 'table' to ForeignKaysTable list (adds data in 'table' field in foreignKeys datatable to list)
Count = 0
For each table in ForeignKeysTable
    Add table to relationships list
    Add ForeignKeysReferencedTable[count] to relationships list
    Count = count + 1

```

Draw Diagram:

This part of the program will draw a diagram of the database with a square representing every table and lines between the tables where there is a primary key foreign key relationship. Also draws a small yellow square on the table where the primary key is located in each relationship.

Sketch design:**Process:**

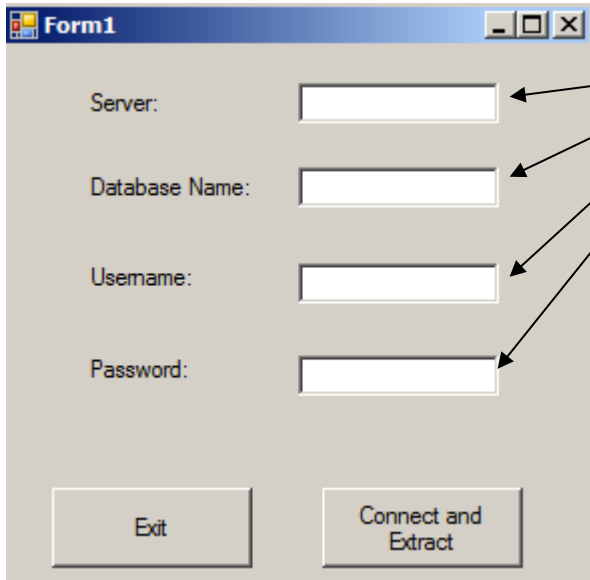
- Iterates through the list of tables and draws a square for each one, with the name of the table written inside each square.
- Each square is 100 pixels by a 100 pixels and a gap of 50 pixels is left in-between each square.
- Draws a maximum of ten squares on each row (will use mod 10 to drop down a line by adding 150 to the y co-ordinate)
- Saves the coordinates it drew each table at into a datatable.
- After it has drawn every table it draws in lines between two tables to represent a relationship, using the datatable with the coordinates of where each table is located. The algorithm will take each table one by one and then check to see if this primary key is a foreign key in a different table (by iterating across all of the tables with foreign keys). *(the midpoint of the x co-ordinates and y co-ordinates will be calculated for each table thus giving the starting and ending point of the line to be drawn)*
- Without the datatable it would not know where all the tables are located, so it would not be able to correctly draw the lines.
- Finally it draws a mini yellow square to represent which of the tables has the primary key in each relationship.

Pseudo code:

```
//draw a square for each table
loop = 0
for each table in TableList
    Create a new rectangle
    x1 = x
    y1 = y
    x2 = x + width {100}
    y2 = y + width {100}
    Draw rectangle
    Put text into rectangle of table name
    loop = loop + 1 {add one to count of tables drawn}
    if loop mod 10 = 0 {if we have drawn a multiple of 10 tables drop a line}
        y = y + 150      {drop down a line}
        x = 0            {reset x back to 0}
    else x = x + 150     {shift across for next rectangle}
    Add rectangle co-ordinates to tablePositions
// draw in relationships
for each referencedTable
    for each table in tablePositions
        if referencedTable name = tablePosition name
            calculate mid-point of rectangle
        if foreignKeyTable name = tablePosition name
            calculate mid-point of rectangle
            draw line between two mid-points
            draw yellow rectangle on first primary table mid-point
```

User Interface:

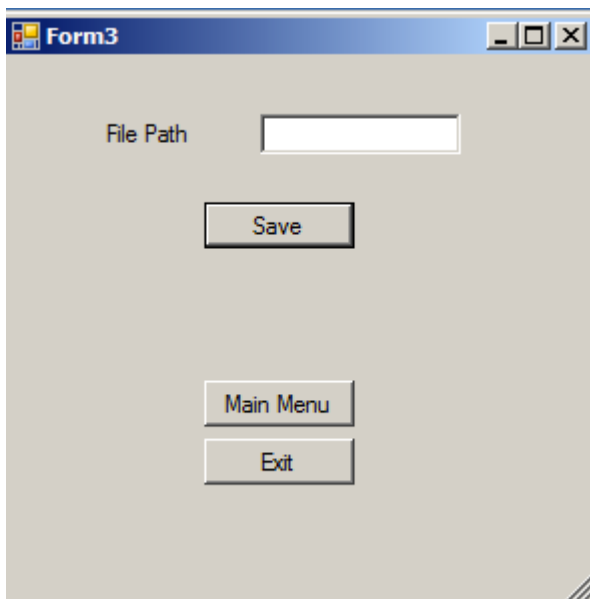
Input Forms Prototyped in Visual Studio using the form tools:



The screenshot shows a window titled 'Form1' with a light gray background. It contains four vertically stacked text input fields. Each field is preceded by a label: 'Server:', 'Database Name:', 'Username:', and 'Password:'. Below the input fields are two buttons: 'Exit' on the left and 'Connect and Extract' on the right. Four arrows point from the text to the right towards each of the four text input boxes.

This form will have text boxes for the user to input their server name, database name and a username and password that will allow access to the database that they want to connect to.

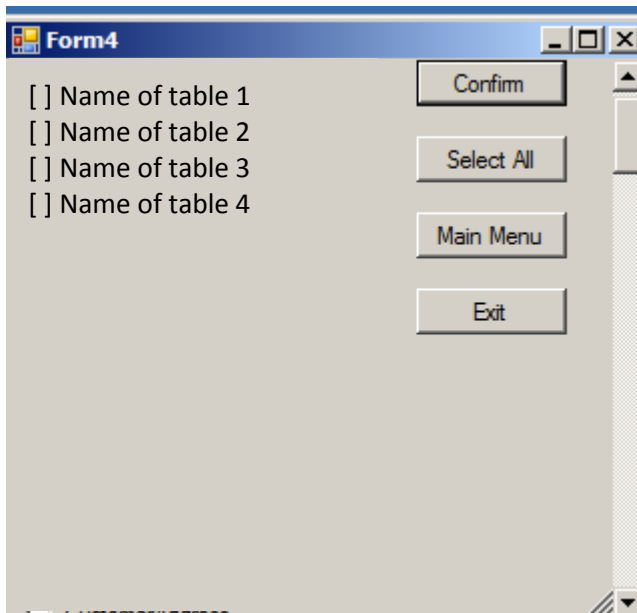
The text boxes are evenly spaced out and have a label next to each one, so that the user does not get confused and they can easily tell what information they are meant to be inputting where.



The screenshot shows a window titled 'Form3' with a light gray background. It features a single text input field labeled 'File Path'. Below the input field is a 'Save' button. Further down are two more buttons: 'Main Menu' and 'Exit'.

This form will allow the user to enter where they would like to save their diagram. They can also leave the textbox blank and then it will save it to a default location, which is on their C: drive.

The forms all have the same style and themes, so that the program has consistency and has a professional look. It also means that the program moves more smoothly from one form to the other.



This form is a mixture between an input form and an output form, as it will show the user a list of all the tables in their database and then asks them to tick which ones they would like to include in their diagram.

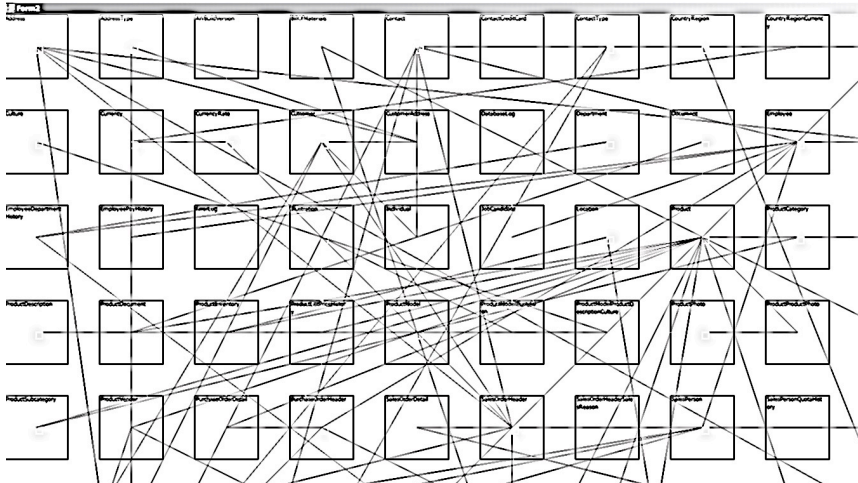
Confirm - moves onto save diagram form

Select all - selects all tables in list and moves onto save diagram form

Main menu - back to the SQL connection form

Exit - quits the program

Outputs:



If the user chooses to just view their diagram rather than save it, then they will see something similar to this.

As discussed earlier it will draw one box per table across the screen (in rows of ten).

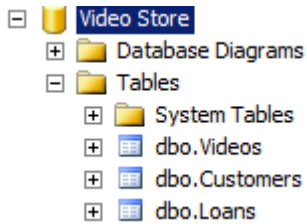
Lines will be drawn between tables to identify relationships and boxes will be drawn on the primary table.

Security:

My program will need to securely connect to a database, so it will need to use user authentication. Depending on the database, this will vary from just being a password to having a username as well or other types of authentication. How secure the ER diagrams are, when saved on the user's computer, will depend on the security of their network and their computer, so it is up to them to ensure appropriate protection. As the ER diagrams are not sensitive data, there is no point in encrypting the data, as it would not matter if someone else saw it. The only thing that would need encrypting would be the logins to the databases in the connection string, but seeing as those strings are only used by the program during runtime and are not being saved to the user's computer, there is no need to encrypt them.

Design Explained Through Detailed Run Through of the Program with an Example Database:

To demonstrate clearly how my program will work, I have created a very simple database called Video Store, shown below. The database has three tables in it, called videos, customers and loans. Each table has a primary key column, called VideoID, CustomerID and LoanID respectively. The loans table has two further columns in it called VideoID and CustomerID, which are foreign keys from the videos and customers tables respectively.



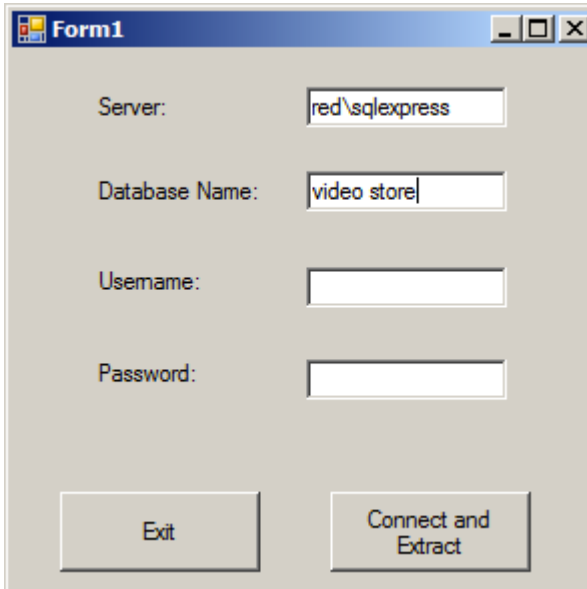
When the tables query and the foreign keys query are run on it in Microsoft SQL Studio, these are the results:

	TableCatalog	TableSchema	TableName	TableDescription
1	Video Store	dbo	Customers	NULL
2	Video Store	dbo	Loans	NULL
3	Video Store	dbo	Videos	NULL

	FK_NAME	schema_name	table	column	referenced_schema	referenced_table	referenced_column
1	FK_Loans_Videos	dbo	Loans	VideoID	dbo	Videos	VideoID
2	FK_Loans_Customers	dbo	Loans	CustomerID	dbo	Customers	CustomerID

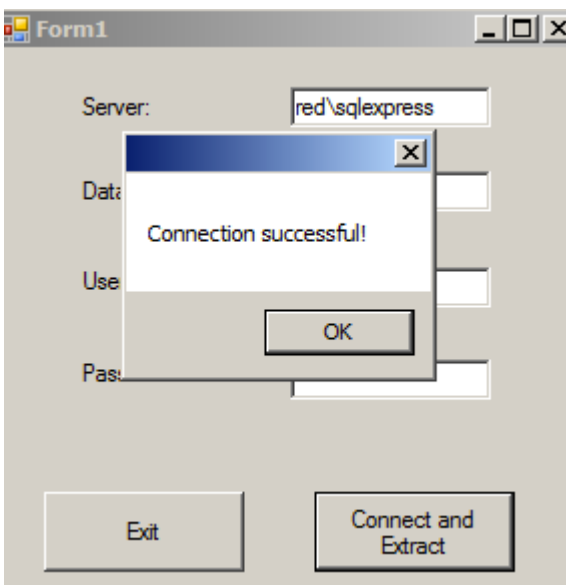
Therefore when my program is run on this database, it should find out that there are only three tables in the database and that there are only two relationships between those tables.

Firstly the details of the database are entered into the first form, which will look like this when it is finished.



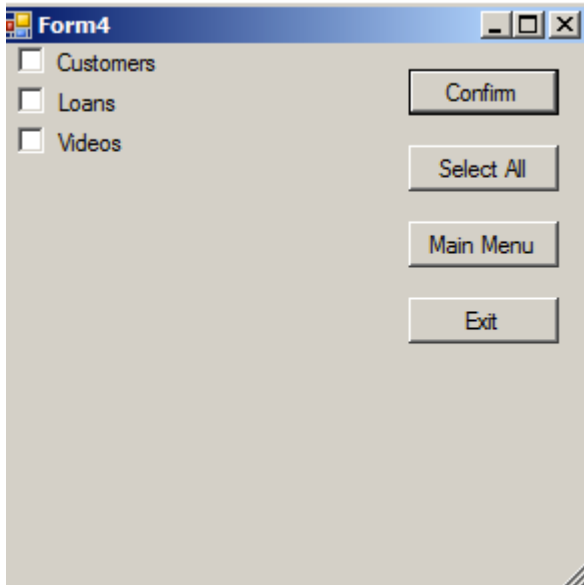
The screenshot shows a Windows application window titled "Form1". It contains four input fields: "Server:" with the text "red\sqlexpress", "Database Name:" with the text "video store", "Username:" which is empty, and "Password:" which is empty. At the bottom of the form, there are two buttons: "Exit" on the left and "Connect and Extract" on the right.

Then the connect and extract button is pressed and the connection successful message will pop up, indicating that the details entered were correct and that the program has managed to connect to the database.

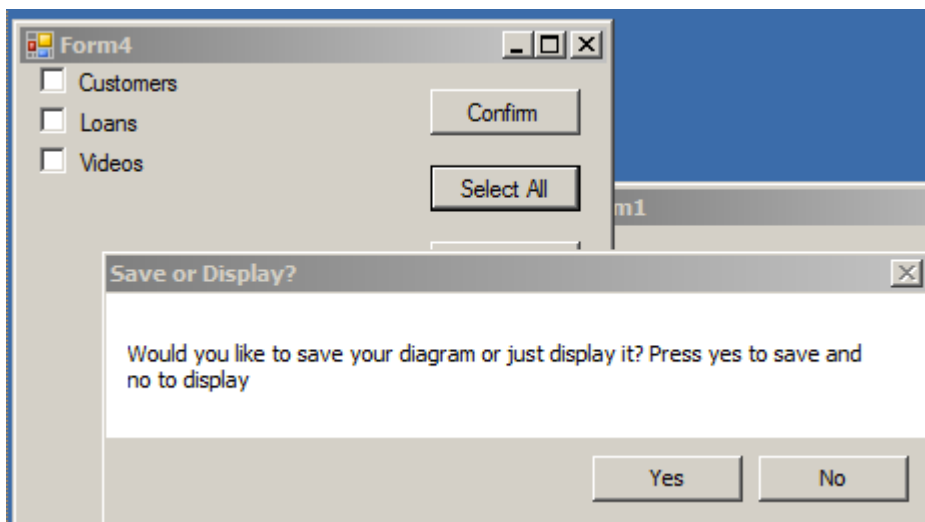


The screenshot shows the same "Form1" window as before, but with a small dialog box overlaid in the center. The dialog box has a title bar with a close button (X) and contains the text "Connection successful!". Below the text is an "OK" button. The background form is partially obscured by the dialog box.

After ok is clicked, the program will run the queries on the database and store the data it retrieves into two datatables. The datatables created will be identical to the two tables created by running the queries in Microsoft SQL Studio. The tableList list would currently have the values Customers, Loans and Videos. This form will then be displayed with a list of the tables.



When select all is clicked, a prompt will be shown, asking whether I would like to save the diagram or just view it. In this example I will choose to just view it by pressing no.

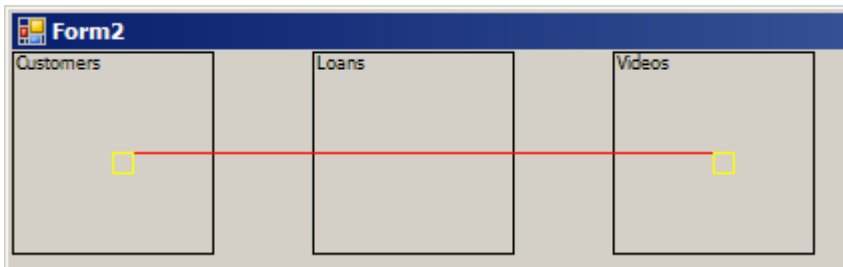


After the button labelled no is clicked, the drawing form will be loaded and because no was clicked instead of yes, the diagram will be drawn to a form instead of to a bitmap. As select all was pressed all of the tables in the database will be used in the drawing, so tableList would still contain Customers, Loans and Videos. A new datatable would then be created to store the location of each table in the diagram, called tablePositions. This datatable would have the following columns; table_name, x1, y1, x2 and y2. The reason why this datatable will be used is so that I can draw a square to represent each table and then draw in the relationships later. The list that will be returned from the getRelationships function would have the following values in it; Loans, Customers, Loans, Videos. This means that there is a foreign key in the Loans table that is the primary key in the Customers table and that there is another foreign key in the Loans table that is a primary key in the Videos table. This list will then be split into two lists called foreignKeyTables, which will have the values; Loans, Loans, and referencedTables, which will have the values; Customers, Videos.

Next the program will loop through each of the referenced tables (the tables whose primary key is used as a foreign key elsewhere) and then find the coordinates of each table using the tablePositions datatable. Next it will loop through each of the foreign key tables (the tables that have a foreign key) until it finds the correct table that matches up with the current referenced table. For example the first referenced table in this example will be Customers and the corresponding value in the foreignKeyTables list is Loans, so the program would get the coordinates of both tables and then a line would be drawn between the midpoints of the two tables. In this example the tablePositions datatable would look like this:

table_name	x1	y1	x2	y2
Customer	0	0	100	100
Loans	150	0	250	100
Videos	300	0	400	100

And the created diagram would look like this:



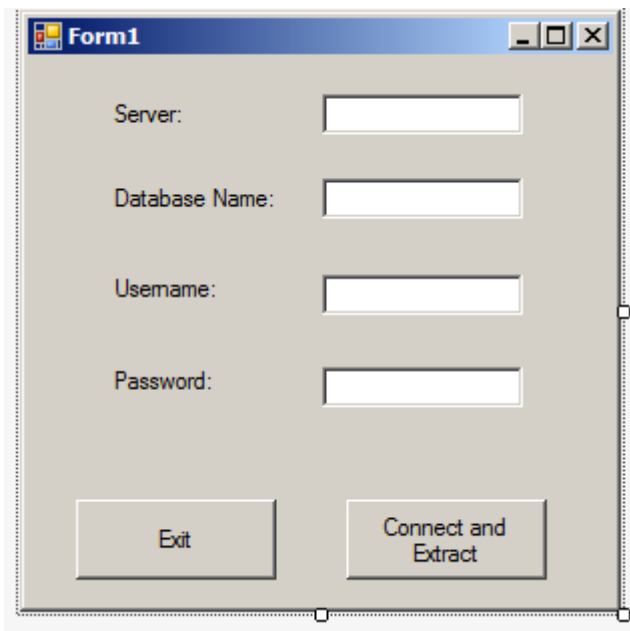
Where there is a line going from Customers to Loans and a line going from Videos to Loans. The yellow squares indicate that the primary keys are located in Customers and Videos.

The dimensions of each table will be 100x100 pixels and there will be a gap of 50 pixels between each table. I will set up the program, so that there will be a max of 10 tables on a row, so after 10 have been drawn, the y coordinate will be incremented by 150, so that a gap of 50 pixels is left between the bottom of each table on one row and the top of each table on the next row. This just means that the diagram will be clearer to read, as the tables are nicely spread out.

Technical Solution

Form 1

Design View:



Code:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Data.SqlClient;

namespace Project
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            this.DoubleBuffered = true;
        }
    }
}
```

```
private void textBox1_TextChanged(object sender, EventArgs e)
{
    // leave empty
}

private void textBox2_TextChanged(object sender, EventArgs e)
{
    // leave empty
}

private void textBox3_TextChanged(object sender, EventArgs e)
{
    // leave empty
}

private void textBox4_TextChanged(object sender, EventArgs e)
{
    // leave empty
}

private void label1_Click(object sender, EventArgs e)
{
    // leave empty
}

private void label2_Click(object sender, EventArgs e)
{
    // leave empty
}

private void button1_Click(object sender, EventArgs e)
{
    // creates boolean for whether to print lots of messages for use in testing to see how the code runs
    bool testing = false;
    // whether connecting to adventureworks for testing, or user database
    bool isAdventureWorks = false;
    // user input from textboxes
    string serverString = textBox1.Text;
    string databaseString = textBox2.Text;
    string usernameString = textBox3.Text;
    string passwordString = textBox4.Text;
    // initialise empty datatables
    DataTable tables = new DataTable();
    DataTable columns = new DataTable();
    DataTable foreignKeys = new DataTable();
    // initialise some lists
    List<string> tableList = new List<string>();
    List<string> foreignKeyList = new List<string>();

    //MessageBox.Show(serverString);
    //MessageBox.Show(databaseString);

    // create connection string for sql connection
    SqlConnectionStringBuilder connectionString = new SqlConnectionStringBuilder();
    if (isAdventureWorks)
```

```

{
    connectionString.DataSource = "red\\sqlexpress";
    connectionString.InitialCatalog = "video store";
    connectionString.IntegratedSecurity = true;
}
else
{
    connectionString.DataSource = serverString;
    connectionString.InitialCatalog = databaseString;
    connectionString.IntegratedSecurity = false;
    connectionString.UserID = usernameString;
    connectionString.Password = passwordString;
}
connectionString.ConnectTimeout = 30;

// use connection string to connect to database
using (SqlConnection myConnection = new SqlConnection(connectionString))
{
    try
    {
        myConnection.Open();
        MessageBox.Show("Connection successful!");
    }
    catch (SqlException ex)
    {
        MessageBox.Show("You failed!" + ex.Message);
    }

    // SQL Queries
    string tablesString = "SELECT TableCatalog = tbl.table_catalog, TableSchema = tbl.table_schema, TableName =
tbl.table_name, TableDescription = prop.value FROM information_schema.tables tbl LEFT JOIN sys.extended_properties
prop ON prop.major_id = object_id(tbl.table_schema + '.' + tbl.table_name) AND prop.minor_id = 0 AND prop.name =
'MS_Description' ORDER BY TableName ASC";
    string foreignKeysString = "SELECT obj.name AS FK_NAME, sch.name AS [schema_name], tab1.name AS
[table], col1.name AS [column], sch2.name AS [referenced_schema], tab2.name AS [referenced_table], col2.name AS
[referenced_column] FROM sys.foreign_key_columns fkc INNER JOIN sys.objects obj ON obj.object_id =
fkc.constraint_object_id INNER JOIN sys.tables tab1 ON tab1.object_id = fkc.parent_object_id INNER JOIN sys.schemas
sch ON tab1.schema_id = sch.schema_id INNER JOIN sys.columns col1 ON col1.column_id = parent_column_id AND
col1.object_id = tab1.object_id INNER JOIN sys.tables tab2 ON tab2.object_id = fkc.referenced_object_id INNER JOIN
sys.schemas sch2 ON tab2.schema_id = sch2.schema_id INNER JOIN sys.columns col2 ON col2.column_id =
referenced_column_id AND col2.object_id = tab2.object_id ORDER BY tab1.name ASC";

    // define an sql command to extract the tables using the tables query
    using (SqlCommand extractTables = new SqlCommand(tablesString, myConnection))
    {
        using (SqlDataAdapter tableReader = new SqlDataAdapter(extractTables))
        {
            try
            {
                // use a data adapter to fill a local datatable with the results of the query
                tableReader.Fill(tables);
                if (testing)
                {
                    MessageBox.Show("Tables extracted");
                }
            }
        }
    }
}

```

```
        catch (InvalidOperationException ex)
        {
            MessageBox.Show("Fail" + ex.Message);
        }
        // list of tables
        foreach (DataRow row in tables.Rows)
        {
            tableList.Add(row.Field<string>("TableName"));
        }
        if (testing)
        {
            MessageBox.Show("Table list created");
        }
    }
}

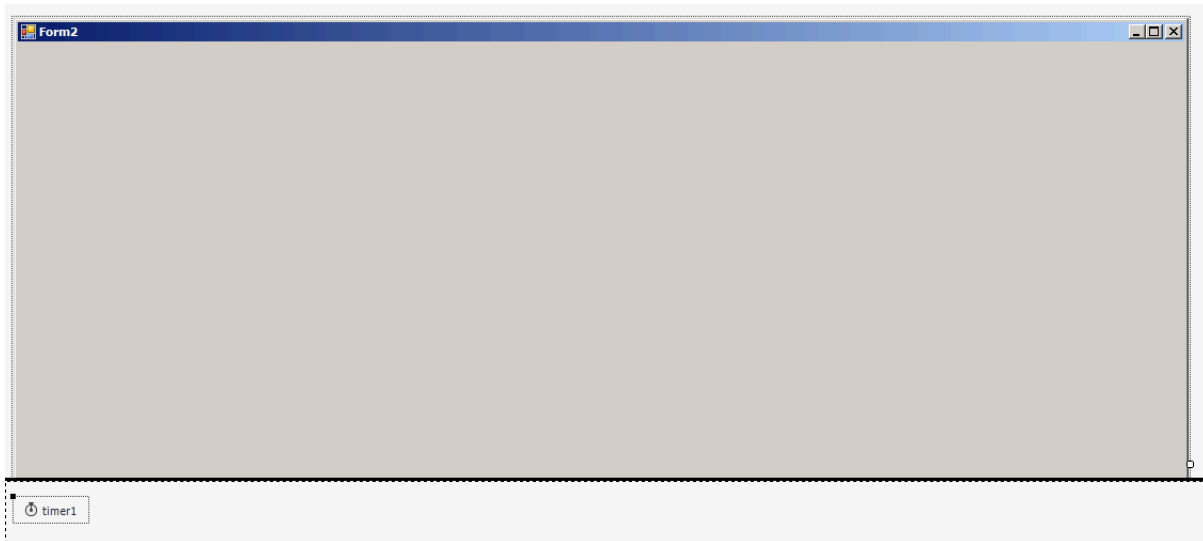
// command to extract info on foreign keys using foreign keys query
using (SqlCommand extractForeignKeys = new SqlCommand(foreignKeysString, myConnection))
{
    using (SqlDataAdapter foreignKeyReader = new SqlDataAdapter(extractForeignKeys))
    {
        try
        {
            // fill in another datatable with results of query
            foreignKeyReader.Fill(foreignKeys);
            if (testing)
            {
                MessageBox.Show("Foreign Keys extracted");
            }
        }
        catch (InvalidOperationException ex)
        {
            MessageBox.Show("Fail" + ex.Message);
        }
        if (testing)
        {
            MessageBox.Show("Foreign key list created");
        }
    }
}

// Get list of tables user wants to display
Form4 chooseTables = new Form4();
chooseTables.tableList = tableList;
chooseTables.tables = tables;
chooseTables.foreignKeys = foreignKeys;
chooseTables.ShowDialog();
}
}

private void button2_Click(object sender, EventArgs e)
{
    Application.Exit();
}
}
}
```

Form 2

Design View:



Code:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Project
{
    public partial class Form2 : Form
    {
        public Form2()
        {
            InitializeComponent();
        }

        public DataTable tables;
        public DataTable foreignKeys;
        public List<string> tableList;
        public bool save;

        private void Form2_Load(object sender, EventArgs e)
        {
            this.DoubleBuffered = true;
            // Sets size of form according to number of tables
            int height = getHeight(tableList);
            this.Width = 1500;
            this.Height = height;
        }
    }
}
```

```
// if user chose to save form, draw diagram as a bitmap
if (save)
{
    this.drawBitmap();
}
// if user chose to display the diagram, draw diagram to a form
else if (save == false)
{
    this.Paint += new PaintEventHandler(DrawingForm_Paint);
}
}

private void drawBitmap()
{
    // datatable to store coordinates of each square representing a table in the diagram
    DataTable tablePositions = new DataTable();
    tablePositions = createDatatable(tablePositions);
    Bitmap bitmap = new Bitmap(this.Width, this.Height);
    Graphics g = Graphics.FromImage(bitmap);
    // give the bitmap a white background
    g.FillRectangle(Brushes.White, 0, 0, this.Width, this.Height);
    // values for drawing the rectangles
    int x = 0;
    int y = 0;
    int height = 100;
    int width = 100;
    // displays more info if testing is true
    bool testing = false;
    if (testing)
    {
        displayList(tableList);
    }
    // get the relationships and store them in a list
    List<string> relationships = getRelationships(foreignKeys, testing);
    List<string> foreignKeyTables = new List<string>();
    List<string> referencedTables = new List<string>();
    int count = 0;
    // get number of tables in database
    int tableListLength = getLength(tableList);
    foreach (string table in relationships)
    {
        if (count == 0 || count % 2 == 0)
        {
            foreignKeyTables.Add(table);
        }
        else
        {
            referencedTables.Add(table);
        }
        count += 1;
    }
    int loops = 0;
    if (save)
    {
        // draw a square for each table
        foreach (string table in tableList)
```

```

{
    Rectangle rectangle1 = new Rectangle(x, y, width, height);
    int x1 = x;
    int y1 = y;
    int x2 = x + width;
    int y2 = y + height;
    g.DrawRectangle(Pens.Black, rectangle1);
    g.DrawString(table, new Font("Tahoma", 7), Brushes.Black, rectangle1);
    DataRow row = tablePositions.NewRow();
    row["table_name"] = table;
    row["x1"] = x1;
    row["y1"] = y1;
    row["x2"] = x2;
    row["y2"] = y2;
    tablePositions.Rows.Add(row);
    loops += 1;
    // max of 10 squares on one row
    if (loops % 10 == 0)
    {
        y += 150;
        x = 0;
    }
    else
    {
        x += 150;
    }
}
int index = 0;
// loops through all the tables where their primary key is used as a foreign key in other tables
foreach (string table in referencedTables)
{
    string foreignKeyTable = foreignKeyTables[index];
    foreach (DataRow row in tablePositions.Rows)
    {
        if (table == row.Field<string>("table_name"))
        {
            int xcoord1 = row.Field<int>("x1");
            int ycoord1 = row.Field<int>("y1");
            int xcoord2 = row.Field<int>("x2");
            int ycoord2 = row.Field<int>("y2");
            float mid1x = (xcoord1 + xcoord2) / 2;
            float mid1y = (ycoord1 + ycoord2) / 2;
            foreach (DataRow row2 in tablePositions.Rows)
            {
                if (foreignKeyTable == row2.Field<string>("table_name"))
                {
                    int xcoord1 = row2.Field<int>("x1");
                    int ycoord1 = row2.Field<int>("y1");
                    int xcoord2 = row2.Field<int>("x2");
                    int ycoord2 = row2.Field<int>("y2");
                    float mid2x = (xcoord1 + xcoord2) / 2;
                    float mid2y = (ycoord1 + ycoord2) / 2;
                    // draws a line connecting the tables to represent a relationship
                    g.DrawLine(Pens.Red, mid1x, mid1y, mid2x, mid2y);
                    // draws a yellow square on the table that has the primary key in the relationship
                    g.DrawRectangle(Pens.Yellow, mid1x, mid1y, 10, 10);
                }
            }
        }
    }
}

```

```

        }
    }
}
index += 1;
}
// loads the save form
Form3 saveForm = new Form3();
saveForm.bitmap = bitmap;
saveForm.ShowDialog();
this.Close();
}
}

```

// function to set the height of the form based on the number of tables in the database

```

private int getHeight(List<string> list)
{
    int height = 0;
    int numberOfTables = getLength(tableList);
    int requiredRows = (numberOfTables / 10) + 1;
    height = requiredRows * 100;
    int numberOfGaps = requiredRows - 1;
    height = height + (numberOfGaps * 50) + 50;
    return height;
}

```

// does the same as drawing the diagram as a bitmap, but draws it to a form instead, which can be immediately displayed to the user

```

private void DrawingForm_Paint(object sender, System.Windows.Forms.PaintEventArgs e)
{
    DataTable tablePositions = new DataTable();
    tablePositions = createDatatable(tablePositions);
    int x = 0;
    int y = 0;
    int height = 100;
    int width = 100;
    bool testing = false;
    if (testing)
    {
        displayList(tableList);
    }
    List<string> relationships = getRelationships(foreignKeys, testing);
    List<string> foreignKeyTables = new List<string>();
    List<string> referencedTables = new List<string>();
    int count = 0;
    int tableListLength = getLength(tableList);
    foreach (string table in relationships)
    {
        if (count == 0 || count % 2 == 0)
        {
            foreignKeyTables.Add(table);
        }
        else
        {
            referencedTables.Add(table);
        }
    }
}

```

```
    count += 1;
}
int loops = 0;
if (save == false)
{
    foreach (string table in tableList)
    {
        Rectangle rectangle1 = new Rectangle(x, y, width, height);
        int x1 = x;
        int y1 = y;
        int x2 = x + width;
        int y2 = y + height;
        e.Graphics.DrawRectangle(Pens.Black, rectangle1);
        e.Graphics.DrawString(table, new Font("Tahoma", 7), Brushes.Black, rectangle1);
        DataRow row = tablePositions.NewRow();
        row["table_name"] = table;
        row["x1"] = x1;
        row["y1"] = y1;
        row["x2"] = x2;
        row["y2"] = y2;
        tablePositions.Rows.Add(row);
        loops += 1;
        if (loops % 10 == 0)
        {
            y += 150;
            x = 0;
        }
        else
        {
            x += 150;
        }
    }
    int index = 0;
    foreach (string table in referencedTables)
    {
        string foreignKeyTable = foreignKeyTables[index];
        foreach (DataRow row in tablePositions.Rows)
        {
            if (table == row.Field<string>("table_name"))
            {
                int xcoord1 = row.Field<int>("x1");
                int ycoord1 = row.Field<int>("y1");
                int xcoord2 = row.Field<int>("x2");
                int ycoord2 = row.Field<int>("y2");
                float mid1x = (xcoord1 + xcoord2) / 2;
                float mid1y = (ycoord1 + ycoord2) / 2;
                foreach (DataRow row2 in tablePositions.Rows)
                {
                    if (foreignKeyTable == row2.Field<string>("table_name"))
                    {
                        int xcoord1 = row2.Field<int>("x1");
                        int ycoord1 = row2.Field<int>("y1");
                        int xcoord2 = row2.Field<int>("x2");
                        int ycoord2 = row2.Field<int>("y2");
                        float mid2x = (xcoord1 + xcoord2) / 2;
                        float mid2y = (ycoord1 + ycoord2) / 2;
                    }
                }
            }
        }
    }
}
```

```

        e.Graphics.DrawLine(Pens.Red, mid1x, mid1y, mid2x, mid2y);
        e.Graphics.DrawRectangle(Pens.Yellow, mid1x, mid1y, 10, 10);
    }
}
}
}
index += 1;
}
}
}
}

```

```

private void timer1_Tick(object sender, EventArgs e)
{
    this.Refresh();
}

```

```

// returns the length of a list
private int getLength(List<string> list)
{
    int length = 0;
    foreach (string item in list)
    {
        length += 1;
    }
    return length;
}

```

```

// displays a list, by printing each element in it
private void displayList(List<string> list)
{
    foreach (string item in list)
    {
        MessageBox.Show(item);
    }
}

```

```

// initialises the tablepositions datatable
private DataTable createDatatable(DataTable table)
{
    table.Clear();
    table.Columns.Add("table_name", typeof(string));
    table.Columns.Add("x1", typeof(int));
    table.Columns.Add("y1", typeof(int));
    table.Columns.Add("x2", typeof(int));
    table.Columns.Add("y2", typeof(int));
    return table;
}

```

// gets relationships by finding the referenced tables in the foreign keys datatable, which was created by the foreign keys query

```

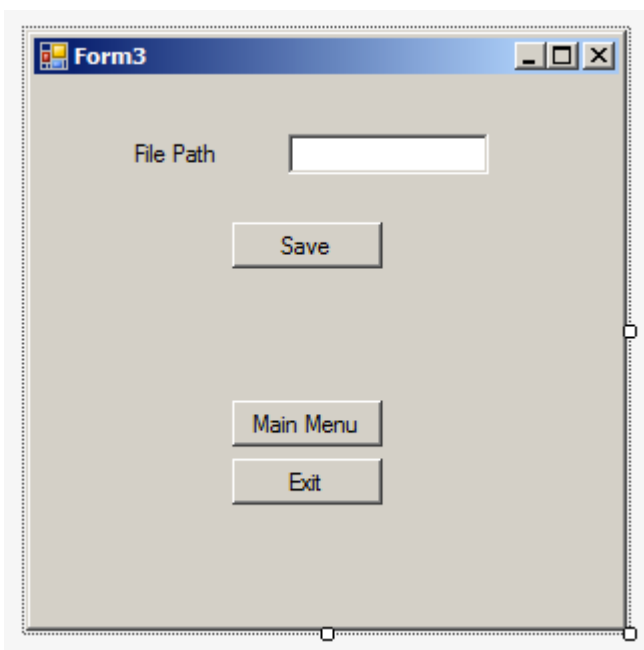
private List<string> getRelationships(DataTable foreignKeys, bool testing)
{
    List<string> ForeignKeys = new List<string>();
    List<string> ForeignKeysReferencedTable = new List<string>();
    List<string> ForeignKeysTable = new List<string>();
    List<string> relationships = new List<string>();
}

```

```
foreach (DataRow row in foreignKeys.Rows)
{
    ForeignKeys.Add(row.Field<string>("FK_NAME"));
    ForeignKeysReferencedTable.Add(row.Field<string>("referenced_table"));
    ForeignKeysTable.Add(row.Field<string>("table"));
}
if (testing)
{
    displayList(ForeignKeys);
    displayList(ForeignKeysReferencedTable);
    displayList(ForeignKeysTable);
}
int count = 0;
foreach (string table in ForeignKeysTable)
{
    relationships.Add(table);
    relationships.Add(ForeignKeysReferencedTable[count]);
    count += 1;
}
if (testing)
{
    displayList(relationships);
}
return relationships;
}
}
```

Form 3:

Design View:



Code:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Drawing.Imaging;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Project
{
    public partial class Form3 : Form
    {
        public Form3()
        {
            InitializeComponent();
        }

        public Bitmap bitmap;

        private void Form3_Load(object sender, EventArgs e)
        {
            this.DoubleBuffered = true;
        }

        private void button1_Click(object sender, EventArgs e)
        {
            // gets filepath from text user entered
            string filePath = textBox1.Text;
            // if it is empty save it to a default location
            if (filePath == "")
            {
                try
                {
                    bitmap.Save("C:/img.jpg", ImageFormat.Jpeg);
                    MessageBox.Show("Image successfully saved!");
                }
                catch (ArgumentNullException ex)
                {
                    MessageBox.Show("Error!" + ex.Message);
                }
                catch (System.Runtime.InteropServices.ExternalException ex)
                {
                    MessageBox.Show("Error!" + ex.Message);
                }
            }
            else
            {
                try
                {
                    bitmap.Save(filePath, ImageFormat.Jpeg);
                    MessageBox.Show("Image successfully saved!");
                }
                catch (ArgumentNullException ex)
                {
                }
            }
        }
    }
}
```

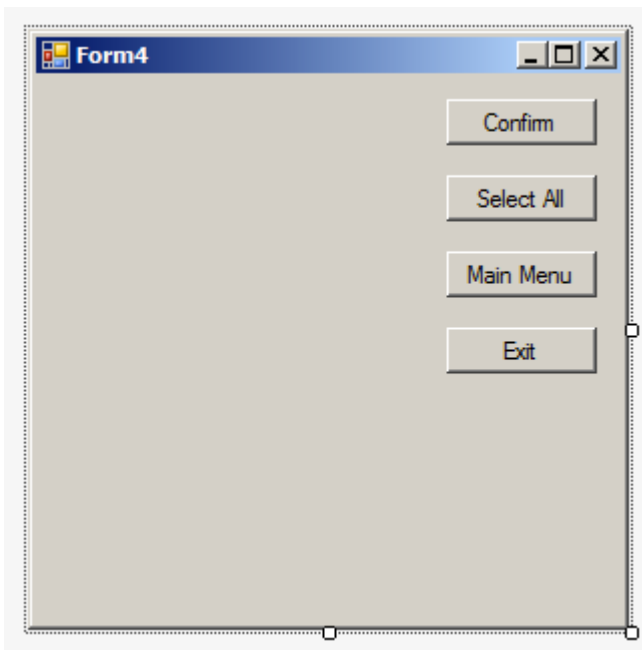
```
        MessageBox.Show("Error!" + ex.Message);
    }
    catch (System.Runtime.InteropServices.ExternalException ex)
    {
        MessageBox.Show("Error!" + ex.Message);
    }
}

private void button2_Click(object sender, EventArgs e)
{
    Application.Exit();
}

private void button3_Click(object sender, EventArgs e)
{
    this.Close();
}
}
}
```

Form 4

Design View:



Code:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```

using System.Windows.Forms;

namespace Project
{
    public partial class Form4 : Form
    {
        public Form4()
        {
            InitializeComponent();

            public DataTable tables;
            public DataTable foreignKeys;
            public List<string> tableList;

            private void Form4_Load(object sender, EventArgs e)
            {
                this.DoubleBuffered = true;
                // sets autoscroll to true so that the form is scrollable, otherwise the user might not be able to see all the tables in
                the database
                this.AutoScroll = true;
                CheckBox box;
                int count = 0;
                // for each table in the database it creates a new checkbox and labels it with the name of the table
                foreach (string table in tableList)
                {
                    box = new CheckBox();
                    box.Tag = table;
                    box.Text = table;
                    box.AutoSize = true;
                    box.Location = new Point(10, count);
                    this.Controls.Add(box);
                    count += 20;
                }
            }

            private void button1_Click(object sender, EventArgs e)
            {
                List<string> newTableList = new List<string>();
                // finds which checkboxes have been ticked by the user, and adds the labels of those checkbox to a list, which is
                then the new list of tables
                foreach (Control control in Controls)
                {
                    if (control is CheckBox)
                    {
                        if (((CheckBox)control).Checked == true)
                        {
                            newTableList.Add(control.Tag.ToString());
                        }
                    }
                }
            }

            // loads the drawing form and prompts the user whether they would like to save or just view the diagram, if they
            would like to save then it sets the save variable as true, otherwise it is false
            Form2 drawingForm = new Form2();
            drawingForm.tables = tables;
            drawingForm.foreignKeys = foreignKeys;
            drawingForm.tableList = newTableList;

```

```
        DialogResult dialogResult1 = MessageBox.Show("Would you like to save your diagram or just display it? Press  
yes to save and no to display", "Save or Display?", MessageBoxButtons.YesNo);  
        if (dialogResult1 == DialogResult.Yes)  
        {  
            drawingForm.save = true;  
        }  
        else if (dialogResult1 == DialogResult.No)  
        {  
            drawingForm.save = false;  
        }  
        drawingForm.ShowDialog();  
        this.Close();  
    }  
  
    private void button2_Click(object sender, EventArgs e)  
    {  
        // if they click select all, then it uses the original table list, including every table  
        Form2 drawingForm = new Form2();  
        drawingForm.tables = tables;  
        drawingForm.foreignKeys = foreignKeys;  
        drawingForm.tableList = tableList;  
        DialogResult dialogResult1 = MessageBox.Show("Would you like to save your diagram or just display it? Press  
yes to save and no to display", "Save or Display?", MessageBoxButtons.YesNo);  
        if (dialogResult1 == DialogResult.Yes)  
        {  
            drawingForm.save = true;  
        }  
        else if (dialogResult1 == DialogResult.No)  
        {  
            drawingForm.save = false;  
        }  
        drawingForm.ShowDialog();  
        this.Close();  
    }  
  
    private void button3_Click(object sender, EventArgs e)  
    {  
        this.Close();  
    }  
  
    private void button4_Click(object sender, EventArgs e)  
    {  
        Application.Exit();  
    }  
}
```

System 'Testing'

Introduction and overview:

Whilst I am programming my project I will be testing each stage on a database called Adventure Works, which is a free open source database, provided by Microsoft. I will be hosting the database locally on a server, so that I can make sure my code is able to successfully connect to a database and so I can test each stage as I make it.

After I have finished creating my program, I will be testing it in three main areas. Firstly I will attempt to connect it to lots of different databases that are hosted on different servers, so that I can see if it has any problems in connecting to certain databases or servers. As I want to make sure that when the program is released it is capable of connecting to whatever database the client want it to.

Secondly I will get it to create ER diagrams of lots of databases to test that it is correctly mapping the foreign keys to their respective primary keys. Preferably I want to be able to get it to draw it in a way that none of the lines and boxes are overlapping, but this will require a lot of programming and testing to make sure that I have done it correctly, therefore I might not have enough time to do this.

The third area that requires testing is to see if it correctly saves the diagram to where the user wants to save it to and also saves it correctly. This will require me to create a lot of different ones and then to save them to my computer, maybe even in different places on the computer, and then to reload them and see if it works correctly.

I have also placed into the Design section a walk-through of the system showing the process from connecting to a database, selecting tables and finally producing an ER diagram. This is cross referenced back to the tables held in the SQL database to show the correctness of this diagram.

Representative Test Table:

Test Number	Description	Data Type	Expected Result	Pass /Fail	Cross Reference
1	Entering correct details of a database and user	Typical	Connection successful message displayed	Pass	Test 1 screenshots
2	Entering incorrect details of database and user	Erroneous	Error message	Pass	Test 2 screenshots
3	Entering correct details of database, but not user	Erroneous	Login failed for user error message	Pass	Test 3 screenshots
4	Clicking exit program button on first form	Typical	Program closes cleanly	Pass	Test 4 screenshots
5	Program correctly creates list of tables and views in database	Typical	Tables in list same as tables and views in database	Pass	Test 5 screenshots
6	Clicking main menu button on choosing tables form	Typical	Goes back to first form	Pass	Test 6 screenshots
7	Clicking exit button on choosing tables form	Typical	Exits program	Pass	Test 7 screenshots
8	Ticking the boxes for some tables and pressing confirm	Typical	Diagram only includes selected tables	Pass	Test 8 screenshots
9	Clicking select all	Typical	Diagram includes all tables	Pass	Test 9 screenshots
10	Choosing yes when prompted	Typical	Brings up save form	Pass	Test 10 screenshots
11	Choosing no when prompted	Typical	Displays diagram	Pass	Test 11 screenshots
12	Clicking save button on save form with no data in textbox	Typical	Saves diagram to default location and success message	Pass	Test 12 screenshots
13	Clicking save button on save form with correct filepath to already existing folder entered in textbox	Typical	Saves diagram to specified location and success message	Pass	Test 13 screenshots
14	Entering filepath to a folder that doesn't already exist and clicking save button on save form	Erroneous	Error message	Fail	Test 14 screenshots
15	Entering invalid filepath on save form and clicking save	Erroneous	Error message	Pass	Test 15 screenshots

16	Pressing main menu on save form	Typical	Goes to first form	Pass	Test 16 screenshots
17	Clicking exit program on save form	Typical	Exits program	Pass	Test 17 screenshots
18	Selecting no tables and clicking confirm on the choose tables form	Erroneous	Creates a blank diagram	Pass	Test 18 screenshots
19	Selecting only one table and clicking confirm on the choose tables form	Typical	Creates a diagram with only one rectangle and no lines	Pass	Test 19 screenshots

Screenshots of Tests:

Test 1:

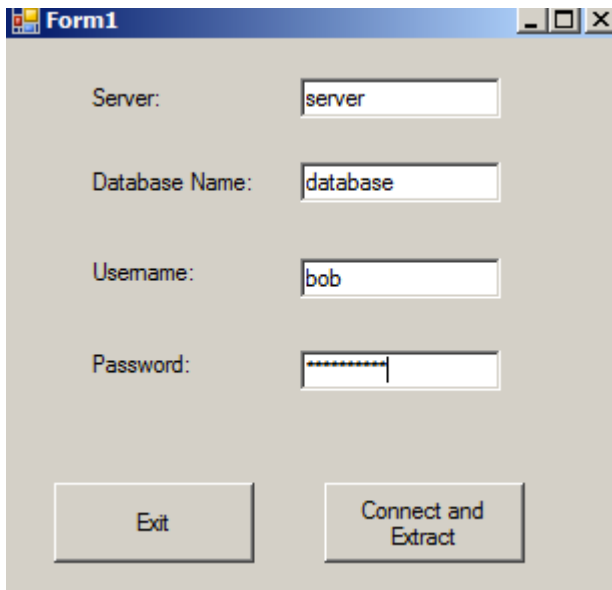
The screenshot shows a Windows-style window titled 'Form1'. It contains four text input fields: 'Server:' with 'red\\sqlexpress', 'Database Name:' with 'adventureworks', 'Username:' with 'Kieran', and 'Password:' with a masked password (seven asterisks). Below the fields are two buttons: 'Exit' and 'Connect and Extract'.

Correct data entered into first form.

This screenshot is similar to the previous one, but a small dialog box is overlaid on top of the main form. The dialog box has a title bar with a close button (X) and contains the text 'Connection successful!' and an 'OK' button. The background form is partially obscured but still visible.

Connection successful message displayed, when connect and extract button is pressed.

Test 2:



Form1

Server:

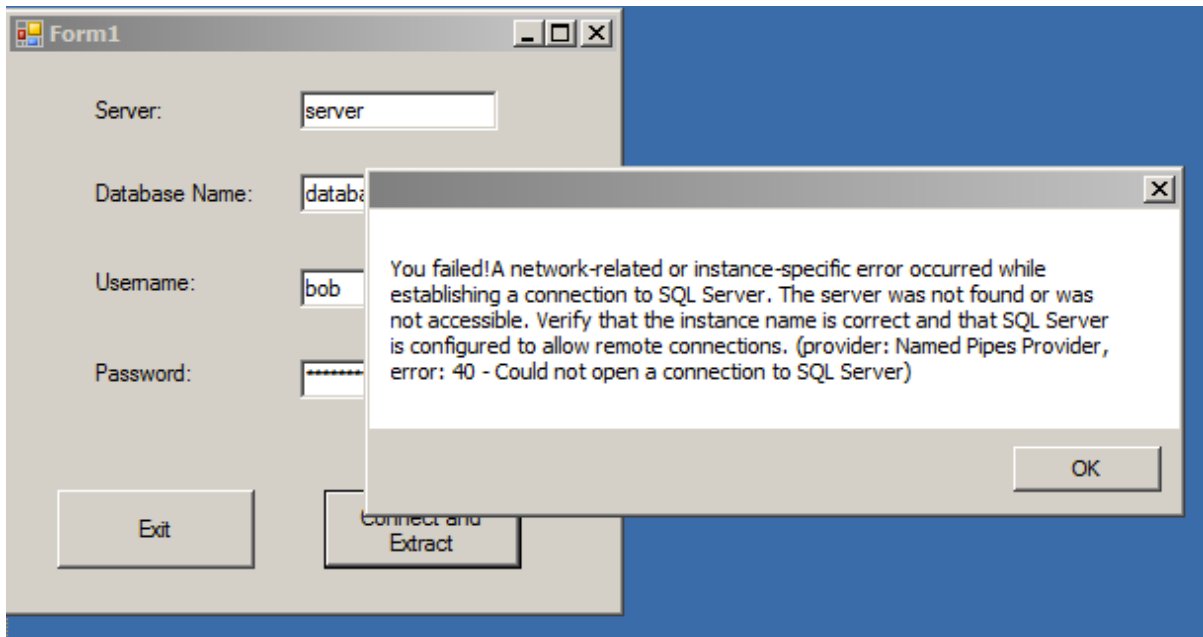
Database Name:

Username:

Password:

Exit Connect and Extract

Incorrect details of a database entered into first form. When connect and extract button is clicked and error message is displayed.



Form1

Server:

Database Name:

Username:

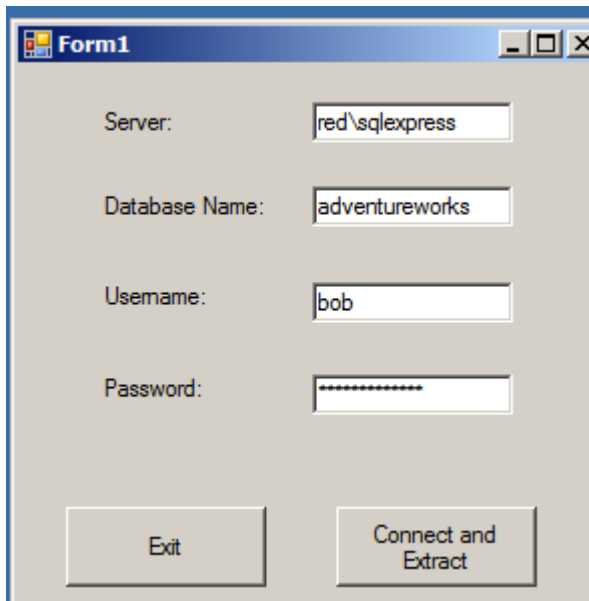
Password:

Exit Connect and Extract

You failed! A network-related or instance-specific error occurred while establishing a connection to SQL Server. The server was not found or was not accessible. Verify that the instance name is correct and that SQL Server is configured to allow remote connections. (provider: Named Pipes Provider, error: 40 - Could not open a connection to SQL Server)

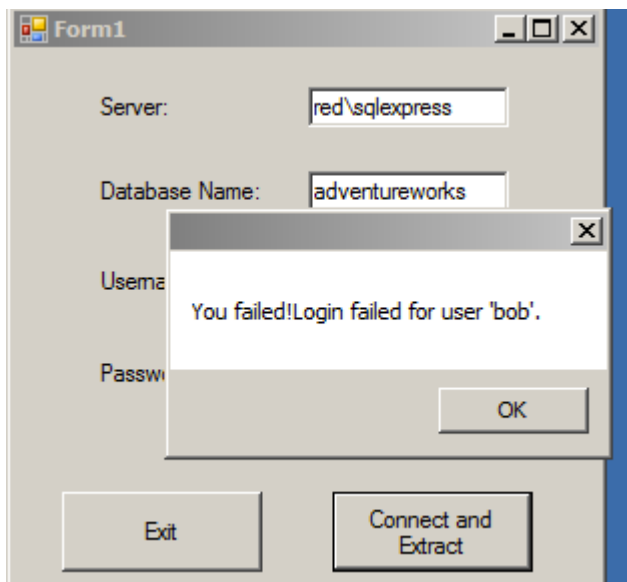
OK

Test 3:



A screenshot of a Windows application window titled "Form1". The window contains four text input fields: "Server:" with the value "red\\sqlexpress", "Database Name:" with the value "adventureworks", "Username:" with the value "bob", and "Password:" with a masked password represented by ten asterisks. At the bottom of the form, there are two buttons: "Exit" on the left and "Connect and Extract" on the right.

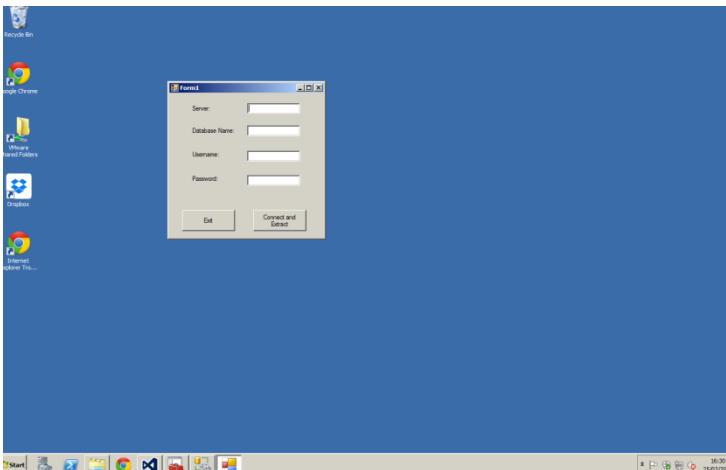
Correct details of database, but invalid login entered into form.



A screenshot of the same "Form1" window. The input fields and buttons are the same as in the previous screenshot. However, a modal error dialog box is now displayed in the foreground. The dialog box has a title bar with a close button and contains the text "You failed!Login failed for user 'bob'." and an "OK" button at the bottom right.

Login failed error message displayed.

Test 4:



Program running, then exit button pressed.



The program exits and is no longer running.

Test 5:

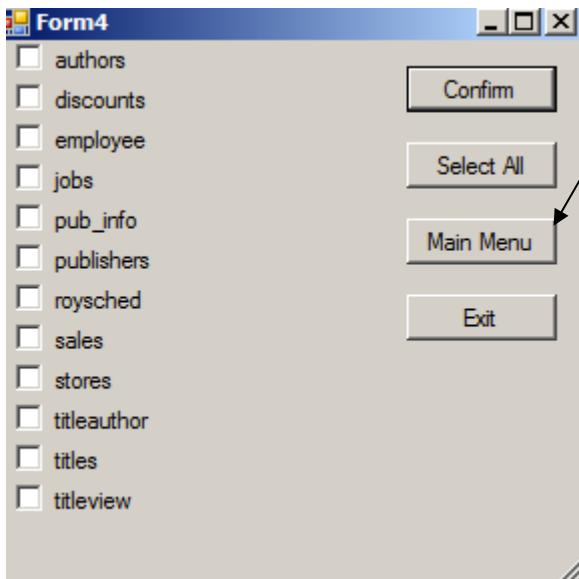
System Tables			
authors	dbo		27/02/2015 09:27
discounts	dbo		27/02/2015 09:27
employee	dbo		27/02/2015 09:27
jobs	dbo		27/02/2015 09:27
pub_info	dbo		27/02/2015 09:27
publishers	dbo		27/02/2015 09:27
roysched	dbo		27/02/2015 09:27
sales	dbo		27/02/2015 09:27
stores	dbo		27/02/2015 09:27
titleauthor	dbo		27/02/2015 09:27
titles	dbo		27/02/2015 09:27

Name	Schema	Create Date	Policy Health State	Owner
System Views				
titleview	dbo	27/02/2015 09:27		dbo

The screenshot shows a Windows application window titled "Form4". On the left side, there is a list of database tables with checkboxes next to each name: authors, discounts, employee, jobs, pub_info, publishers, roysched, sales, stores, titleauthor, titles, and titleview. On the right side, there are four buttons: "Confirm", "Select All", "Main Menu", and "Exit".

All the names in the list match the names in the database shown above.

Test 6:

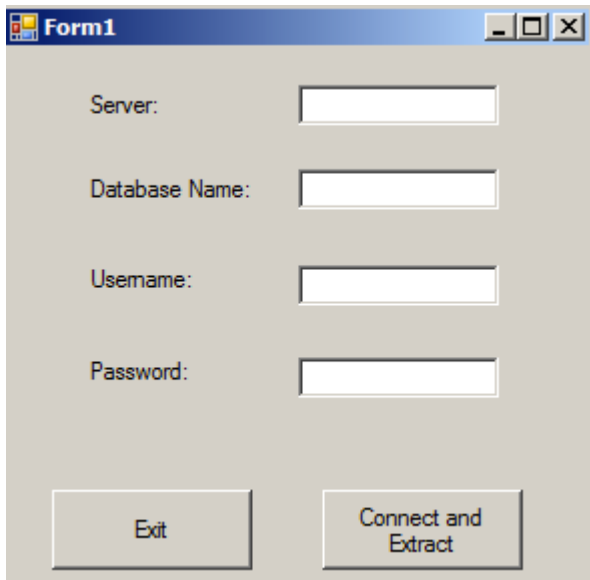


Form4

- authors
- discounts
- employee
- jobs
- pub_info
- publishers
- roysched
- sales
- stores
- titleauthor
- titles
- titleview

Buttons: Confirm, Select All, Main Menu, Exit

Clicking main menu button on choosing tables form.



Form1

Server:

Database Name:

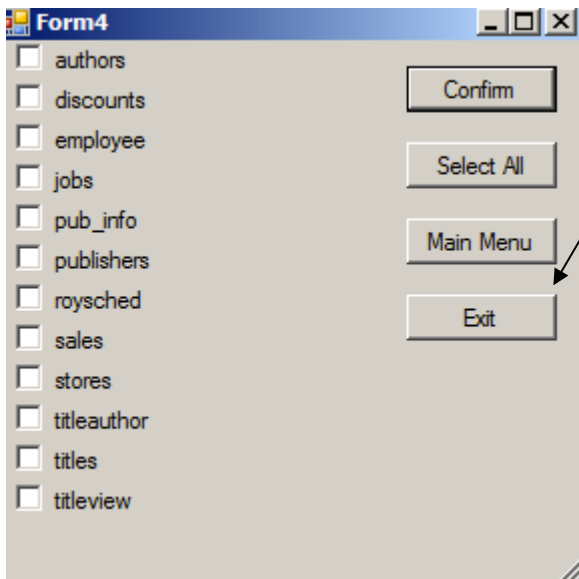
Username:

Password:

Buttons: Exit, Connect and Extract

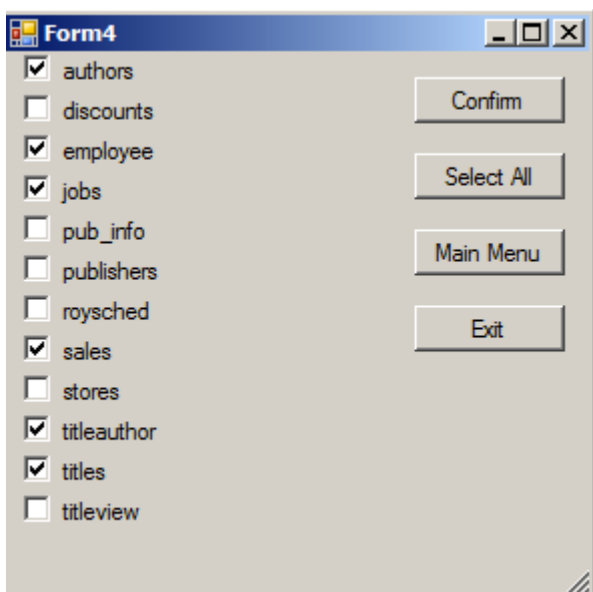
Goes back to first form

Test 7:

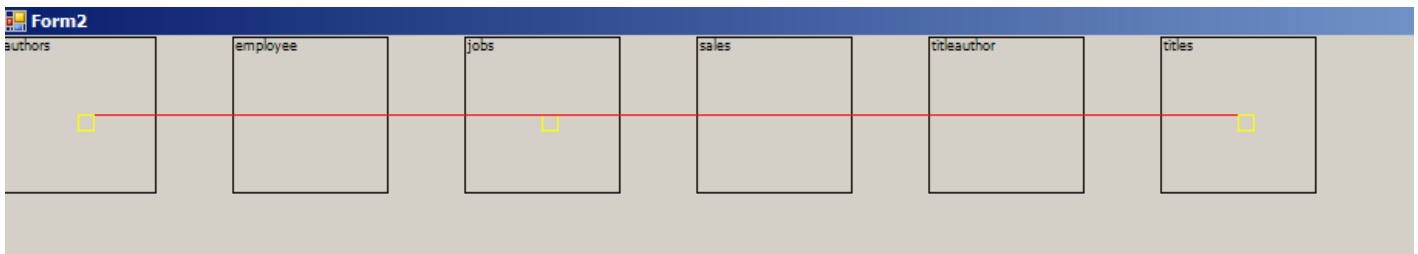


Clicking exit button on choosing tables form safely closes the program.

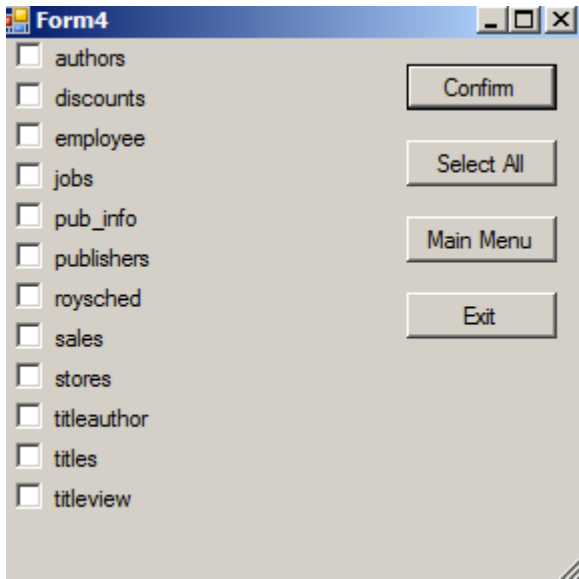
Test 8:



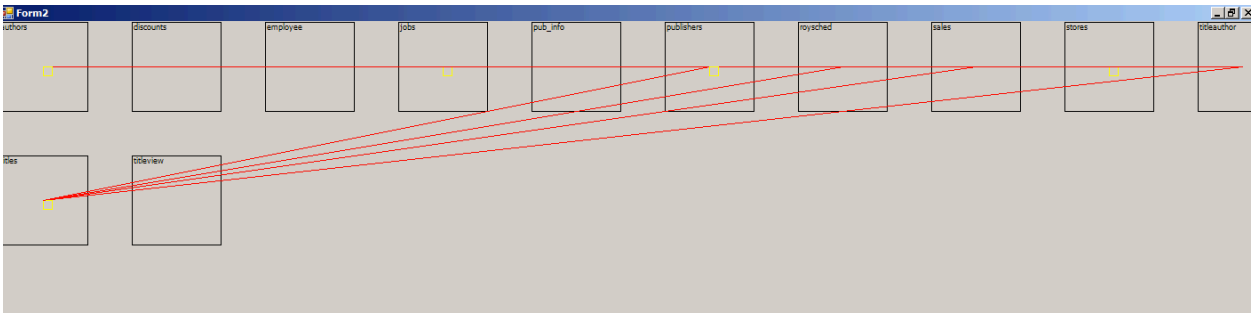
Only some tables are selected then, the confirm button is pressed. Created diagram only includes the tables that were selected.



Test 9:

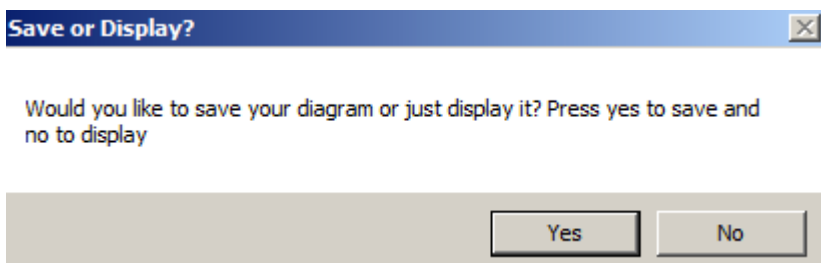


The select all button is pressed.

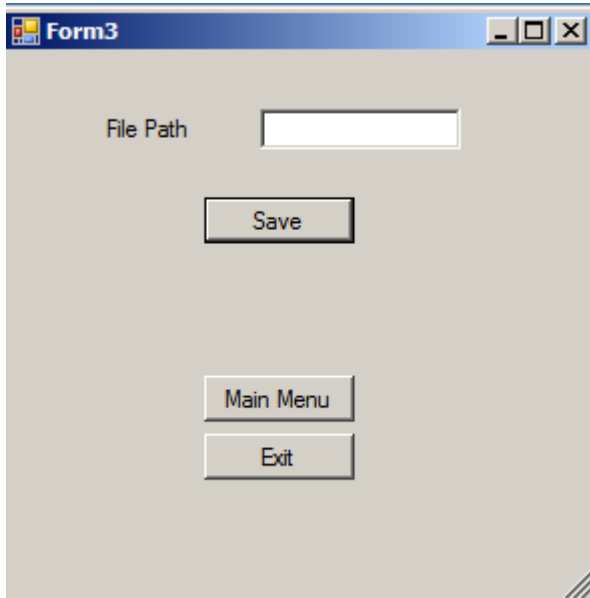


Every table is included in the diagram.

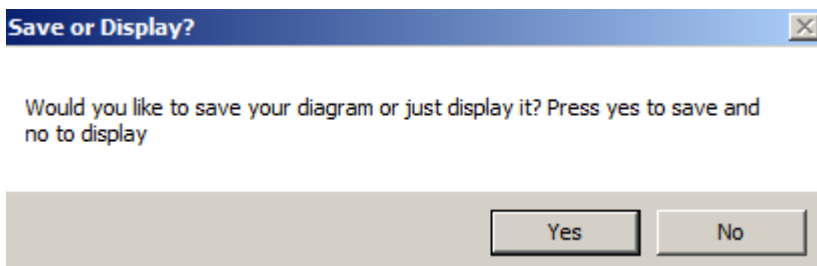
Test 10:



Responding to the prompt by clicking yes brings up the save form.



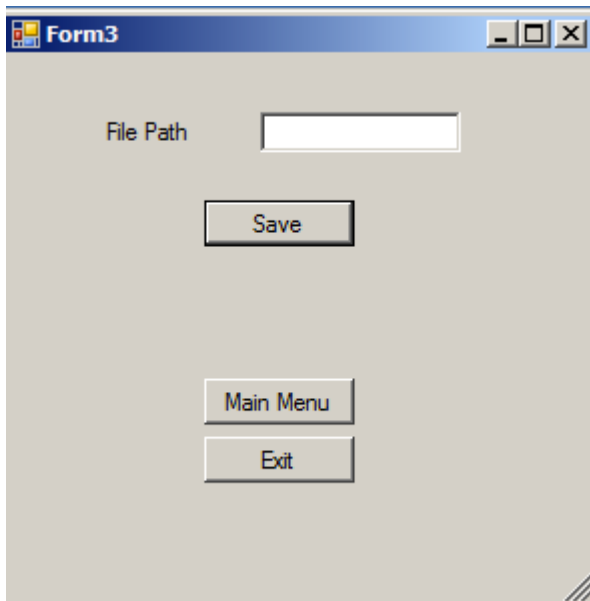
Test 11:



Clicking no just displays the diagram.



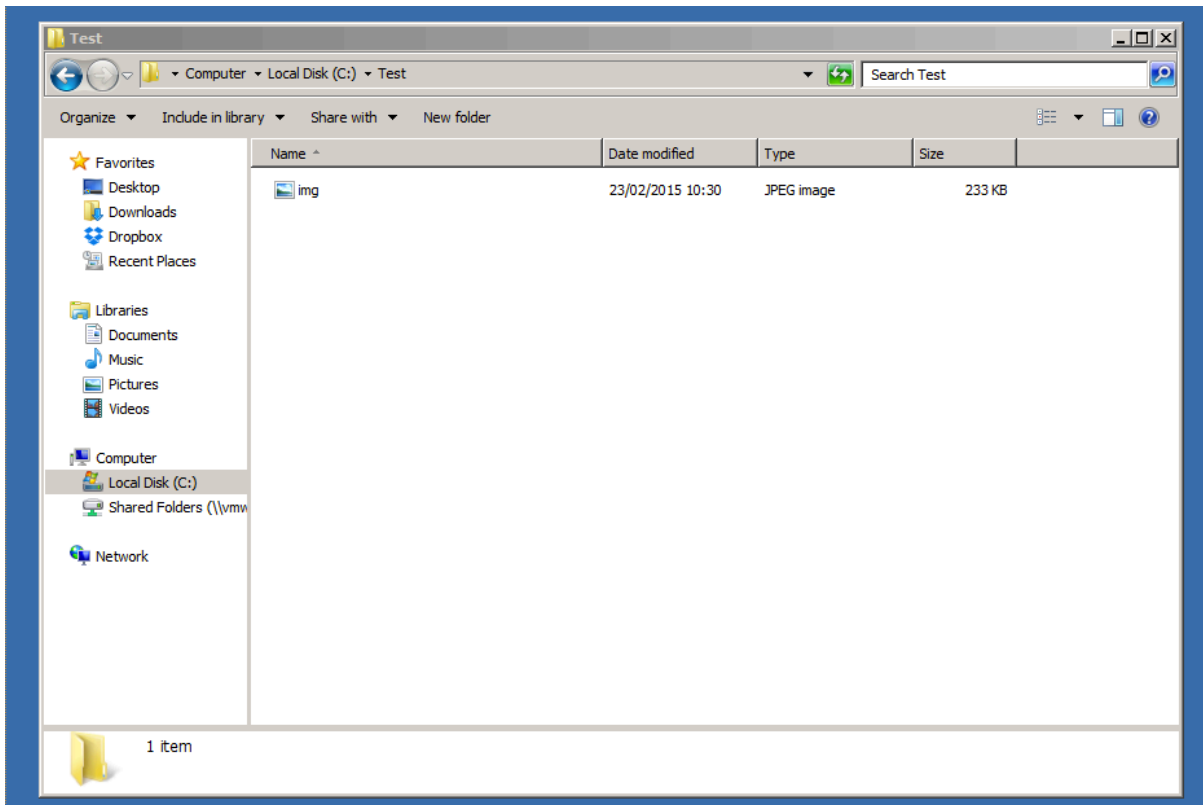
Test 12:



Empty save form with no data entered by user. When the save button is pressed it will save the diagram to the default location, which is on the C drive.

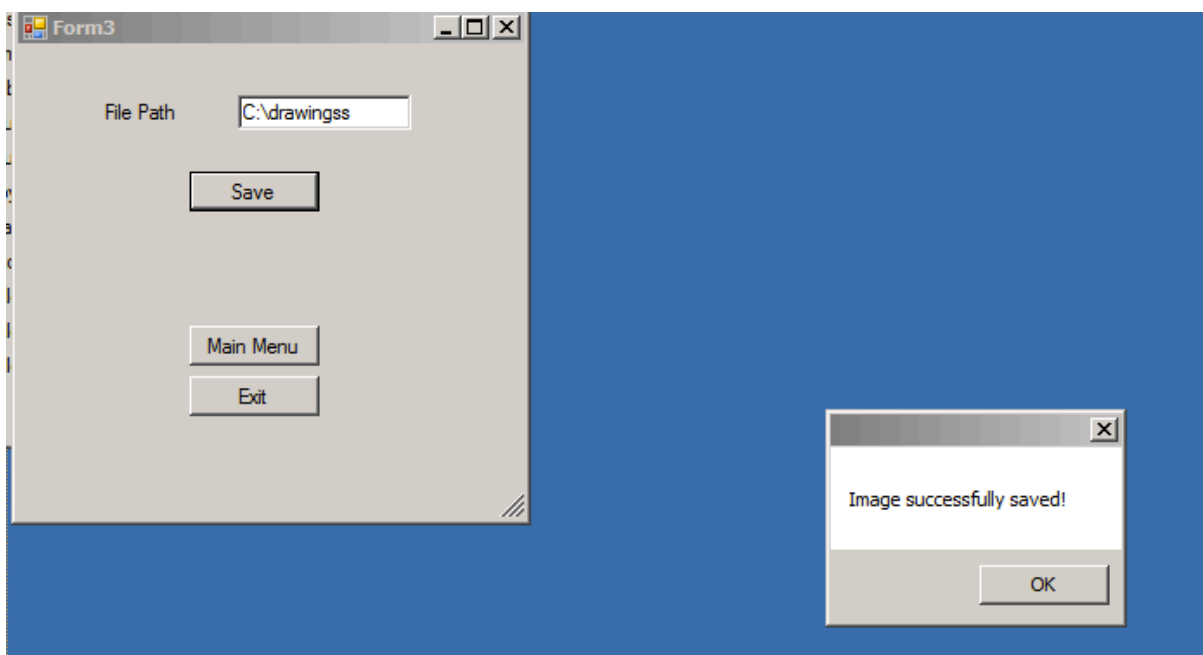
Name ^	Date modified	Type	Size
PerfLogs	14/07/2009 04:20	File folder	
Program Files	23/02/2015 10:56	File folder	
Program Files (x86)	07/02/2015 05:25	File folder	
SQL Server 2000 Sample Databases	27/02/2015 09:18	File folder	
Test	23/02/2015 10:30	File folder	
Users	13/09/2014 14:08	File folder	
Windows	24/03/2015 15:17	File folder	
diagrams	25/03/2015 13:15	File	271 KB
img	26/02/2015 15:56	JPEG image	113 KB
lol	25/03/2015 16:58	File	52 KB

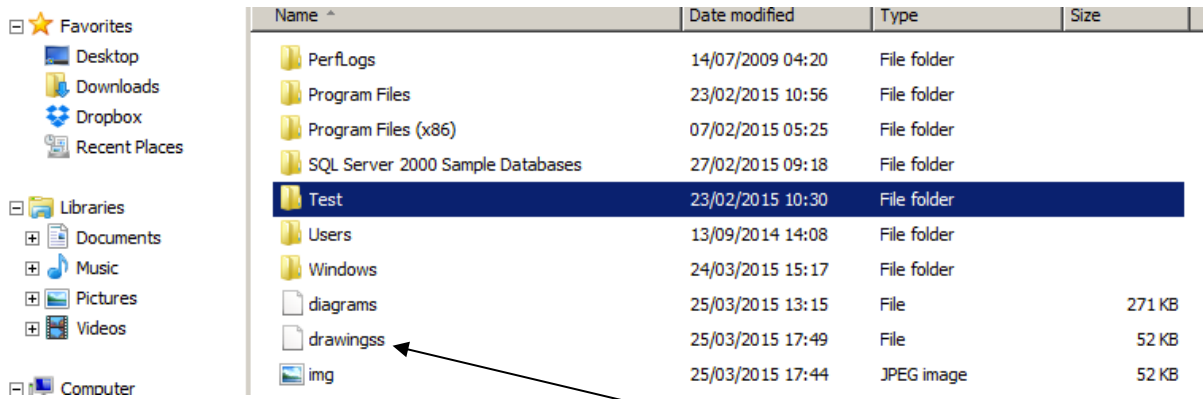
Test 13:



The diagram is correctly saved to the folder called test.

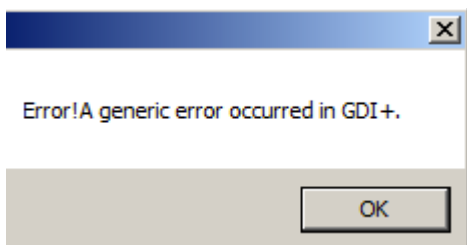
Test 14:





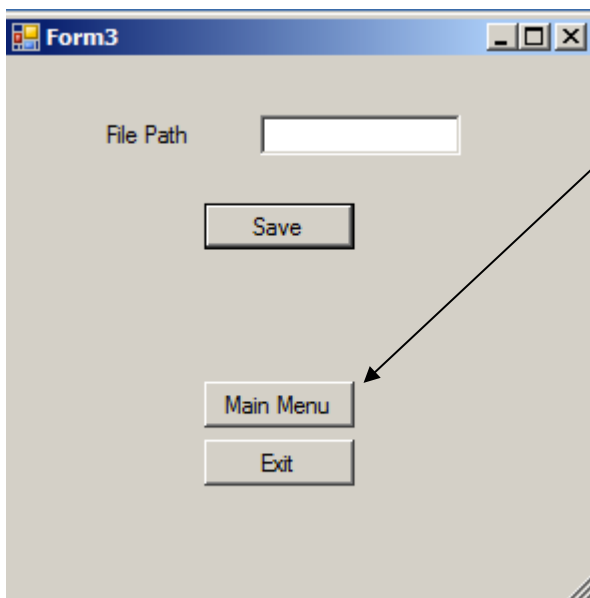
Instead of displaying an error message, the program saves an unreadable file straight onto the C drive.

Test 15:

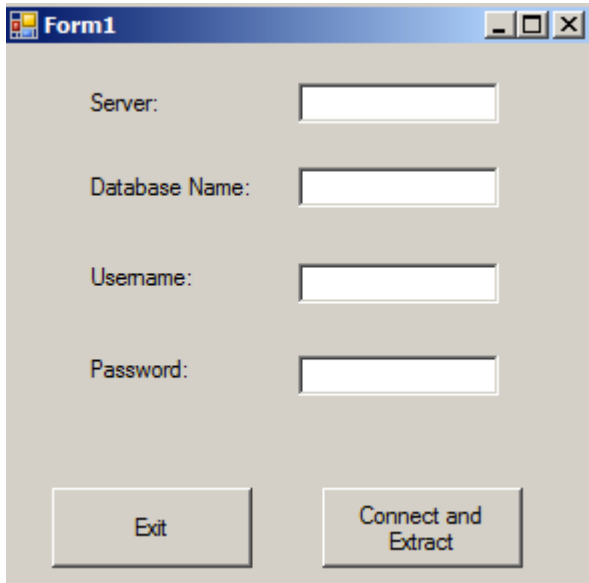


An error message is displayed

Test 16:

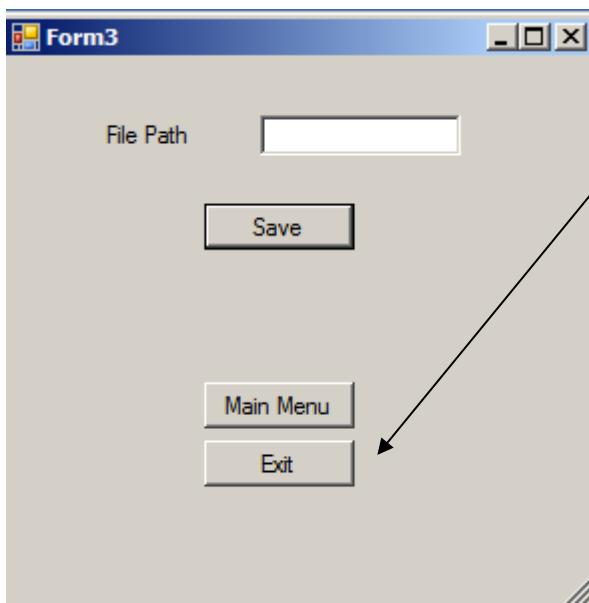


Pressing main menu on the save form takes you back to the first form.



A screenshot of a Windows application window titled "Form1". The window has a standard Windows title bar with minimize, maximize, and close buttons. The main area is light gray and contains four input fields, each with a label to its left: "Server:", "Database Name:", "Username:", and "Password:". Below the input fields are two buttons: "Exit" on the left and "Connect and Extract" on the right.

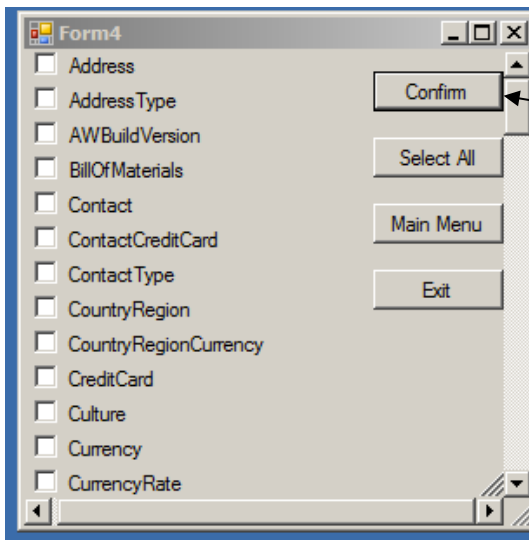
Test 17:



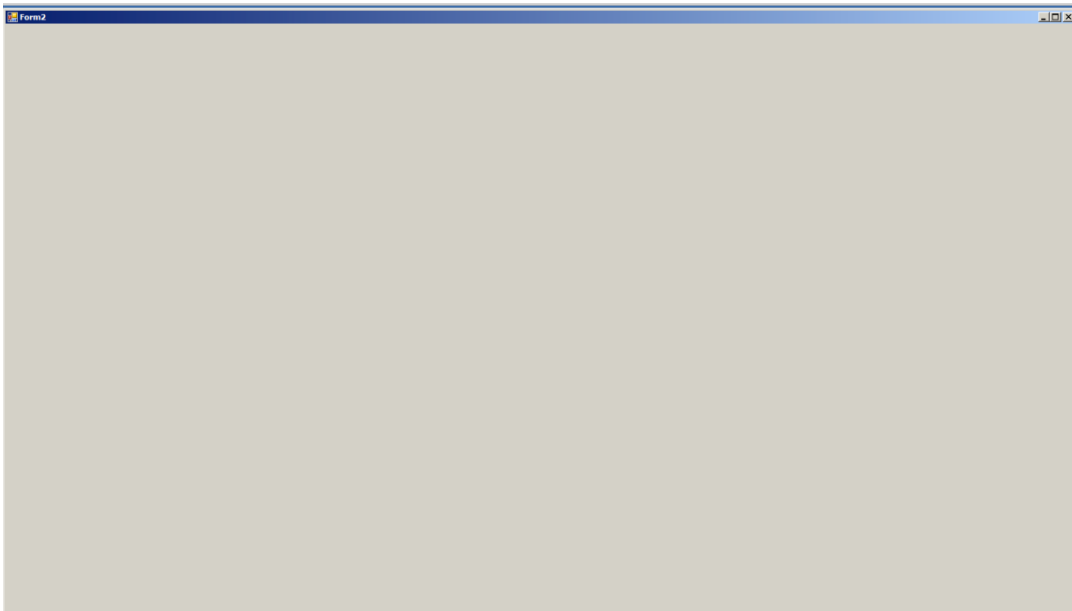
A screenshot of a Windows application window titled "Form3". The window has a standard Windows title bar with minimize, maximize, and close buttons. The main area is light gray and contains a "File Path" label followed by an input field. Below the input field is a "Save" button. Further down are three buttons stacked vertically: "Main Menu", "Exit", and another "Exit" button. An arrow points from the text "Clicking the exit button on the save form exits the program." to the bottom "Exit" button.

Clicking the exit button on the save form exits the program.

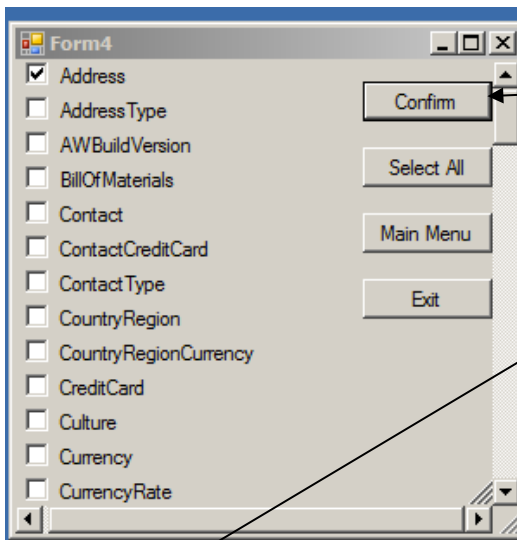
Test 18:



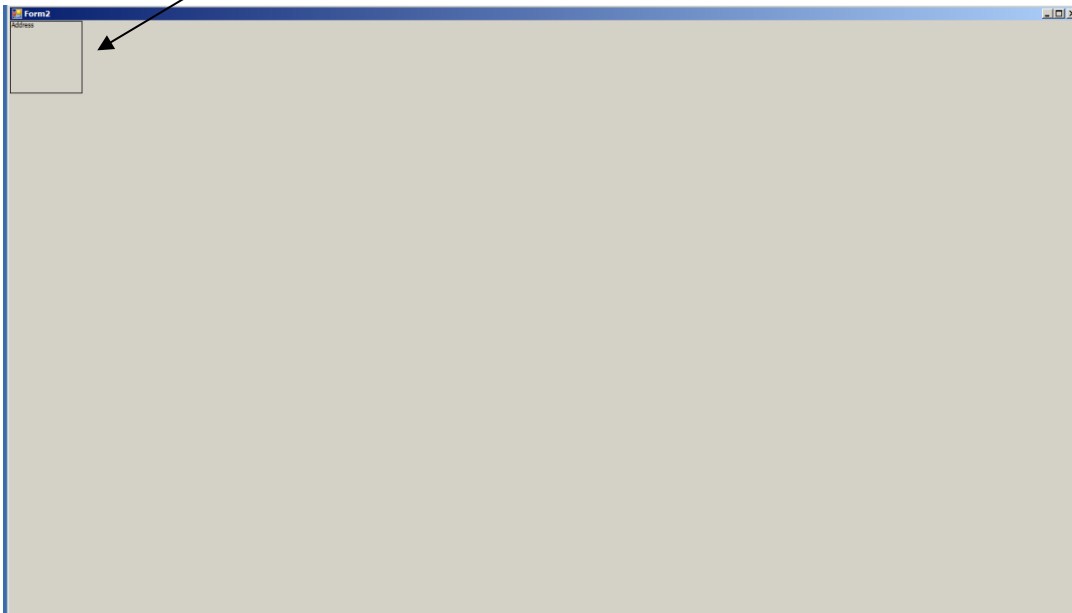
No tables are selected, and then the confirm button is pressed. An empty diagram, as shown below, is then created.



Test 19:



Only one table is selected, and then confirm is clicked. A diagram is then created with only one table and no relationships, as shown below.



Evaluation

Objective Analysis

Objective	Met?	Comment
Allow the user to enter configurations to connect to a remote database of their choice	Yes	Upon launching the program the user will be met with a form where they can enter in the details of the database they want to connect to. They can input the server name, the database name, a username and a password
Output an error message if the program fails to connect to the database, alerting the user that they might have entered the wrong details	Yes	After entering the details and clicking the button labelled connect and extract, if program is unable to connect to the database using the details that the user provided, then it will display an error message
Use SQL queries to extract the metadata and keys from all the different tables and then store that data locally in a C# program written in Visual Studio.	Yes	If the connection is successful, then two SQL queries will be run on the database and the results of the queries will be stored in two local datatables during runtime
Allow the user to choose which tables they want to include in the diagram	Yes	After the connection is successful and the required data has been extracted from the database, the user is presented a list of all of the tables in the database and they can choose which ones they want included in the diagram
Use the data from the database to draw a unique ER diagram of the database	Yes	A diagram is drawn, using the tables that the user has selected
Give the user the option to save the diagram	Yes	If the user chooses to they can save the diagram to a folder of their choice
Refine ER diagram drawing algorithm, so that the diagram is more clear and easy to read (i.e. there are not so many lines crossing over each other)	No	This was very hard to program and I ran out of time, so it have not been included in the final project, but it is a good idea for a possible extension

User Feedback:

How easy is the system to use?

I found the system easy to use and deploy onto my machine. Connection to the target database was easy, although I managed to mistype the server address and the system presented me with an explanatory message. The first time I used it, I was connecting to a very large Database but only interested in a small subset of tables, so the feature to restrict the process and diagram to required tables was very useful. As this is a technical system, I do not foresee it being used by other types of users as it provides a graphical view of a very technically complex environment and even the diagram still requires specialist skills to understand and appraise.

How does the system meet their objectives?

The system delivers what was specified and required. Technically I find it very good and it easily passed the test criteria I asked to be demonstrated. Interestingly for me, I asked for a few test cases based on the AdventureWorks DB from Microsoft and in one case the solution did not deliver the expected results. However, when we examined the results in detail it turned out that my expected results for the test case were incorrect and the system was correct!

How easy was the system to set up?

The User Guide was very useful and I found it easy to follow. It clearly outlined the steps to follow and the download and install process worked exactly as described in the documentation. I used a Windows 2008 R2 Server VM to perform the installation and was pleasantly surprised there were no installation issues.

Do you have any criticisms?

As I am not of the Internet generation I find it hard to be overly critical of free software. That said, if I were to pretend this was a software product I had paid for then I would perhaps be critical of a couple of things. Firstly, although the product performs exactly as prescribed, it is still quite difficult to understand when many tables are presented. It would be much better if the diagram was laid out with related tables clustered together. However, I do understand that this is a reduced travelling salesman problem, therefore difficult to code. Additionally, I find the basic format of the Entities (i.e. squares) and Relationships (i.e. lines) to be not to my personal taste. Entity Relationship diagrams can be shown in many notations and I would prefer the "crowsfoot" notation so that cardinalities and optionalities are clear. As a final point I'd like the GUI to provide more options in terms of colours, fonts, sizes, flexibility etc - but given the time taken to develop the solution and the fact it is 100% correct in technical terms I do feel this is a very harsh criticism.

What improvements or extensions would you recommend?

In the perfect world I would like this system to be a free version of Visio! That said, as a free alternative it offers a significant time-saving over the very manually intensive procedure it replaces. A major element of the Data Migration process is the understanding of the source (or old)

system and often there are scarce resources available about the existing system. Improvements I think would be mainly centred around the GUI interface and the information shown, for example it would be nice to add column details into the Entity boxes and allow drag and drop of objects in the diagram. As a customer I am sufficiently impressed by the solution delivered to request some enhancements be made to the program - on a paid basis!

Analysis of User Feedback:

The feedback that I have retrieved from my main end user is very positive and has made me confident in the fact that I have successfully met all the objectives that I set myself and the requirements that the end user requested. It is also very reassuring to hear that they had no problems installing the software and using it, which means that I was successful in programming the software to be very user friendly and as simple as possible without losing any functionality.

There are however several things that the user did not like about the software, or things that the user would like to be included in the software, such as customisation of the diagram after it has been drawn and also for the diagram to have more detail on it in the first place.

Possible Extensions:

If I had had more time to do this project or I was to do it again, then there are several things that I would do differently. The first things that I would attempt to do would be to add all the extra features that my main end user would like in the software. One of these is to give the user more customisation over their diagram, so that they would be able to choose which notation of Entity Relationship diagrams is used in the diagram and to give them the option of changing the colours or even size of the lines and rectangles in the diagram. I would also add more details to the diagram, so instead of just saying the name of the table for each rectangle, it would also have all the columns of that table, indicating which one was the primary key and which ones are foreign keys, if any.

Another extension that I would do is something I planned on doing from the start, but decided against it as my project got underway, as I realised it would take too much time to do properly and if I put all my effort on doing that then other areas of the project would have suffered. The thing I was planning on doing was making the drawing algorithm better, so that the diagrams were easier to look at and interpret. With the current state of the program you can easily get diagrams which have lines crossing over each other and sometimes it is actually impossible to tell which rectangle a line originated. Also, as my main end user stated, the tables in the diagram are not grouped together in any special way currently, they are simply displayed in alphabetical order. The diagrams would be much better if similar tables were grouped together, or if the tables were placed in a way that meant that the lines representing relationships didn't have to cross over each other. So if I was to spend any more time working on this project, then this would be the first thing that I would be looking to do, but it would probably mean having to re-write large sections of my code.

Currently if the user wants to connect to the same database that they have previously connected to, they have to go back to the first form and enter in all of the details again. A way to fix this problem would be to save the details that the user enters in, so that when they next want to make a connection they can just select one from a drop down list of previous connections, rather than having to type it all out again. The connections would have to be stored locally on the user's computer, possibly to a text file or something similar, and the data would have to be encrypted, as it would be very sensitive data because it would contain information on how to connect to several different databases and if it was not encrypted someone else could access that data and then they would be able to access those databases.

At the moment the queries count views in a database as tables, so when they are extracting a list of all the tables in a database, the created list includes any views that are also in the database. A user might not want the views to be included in the diagram, so instead of them having to manually select all the tables other than the views, I could add in a prompt that asks them whether they would like the views to be included or not. This would mean that I would have two versions of queries, one that includes views and one that does not.

The way that the forms are named currently could be confusing to the user, as the first form is called Form 1, but the second form that the user sees, the choose tables form, is called Form 4. This is because this was the last thing I added to the program, so when I made the form in the code it was called Form 4 by default, as forms 1-3 had already been created. To avoid confusing the user, it would probably be better to name the forms in ascending order, so that the first form the user sees is Form 1 and then Form 2 and so on. Or it would make it even clearer if I gave each form a separate name, such as calling Form 1 the Main Menu, calling Form 4 the Choose Tables Form, calling Form 3 the Save Form and calling Form 2 the Drawing Form.

One final thing that I would like to change about my program is that I would like to change how the program asks the user whether they would like to save their diagram or not. Currently it only allows them to save it and not have a look at the diagram, or have a look at the diagram and not save it. If the user wants to do both then they would have to run the program twice, using the same database. When I was coding this bit initially I tried to do it so that the user would be able to do both in one instance of running the program, but I could not get it to work without errors, so I decided to do it the way it is currently, as it works perfectly fine, so that I could then move onto the next bit of the project. If I had more time then I would have spent longer looking at this problem and then I would have most likely found a way around it.