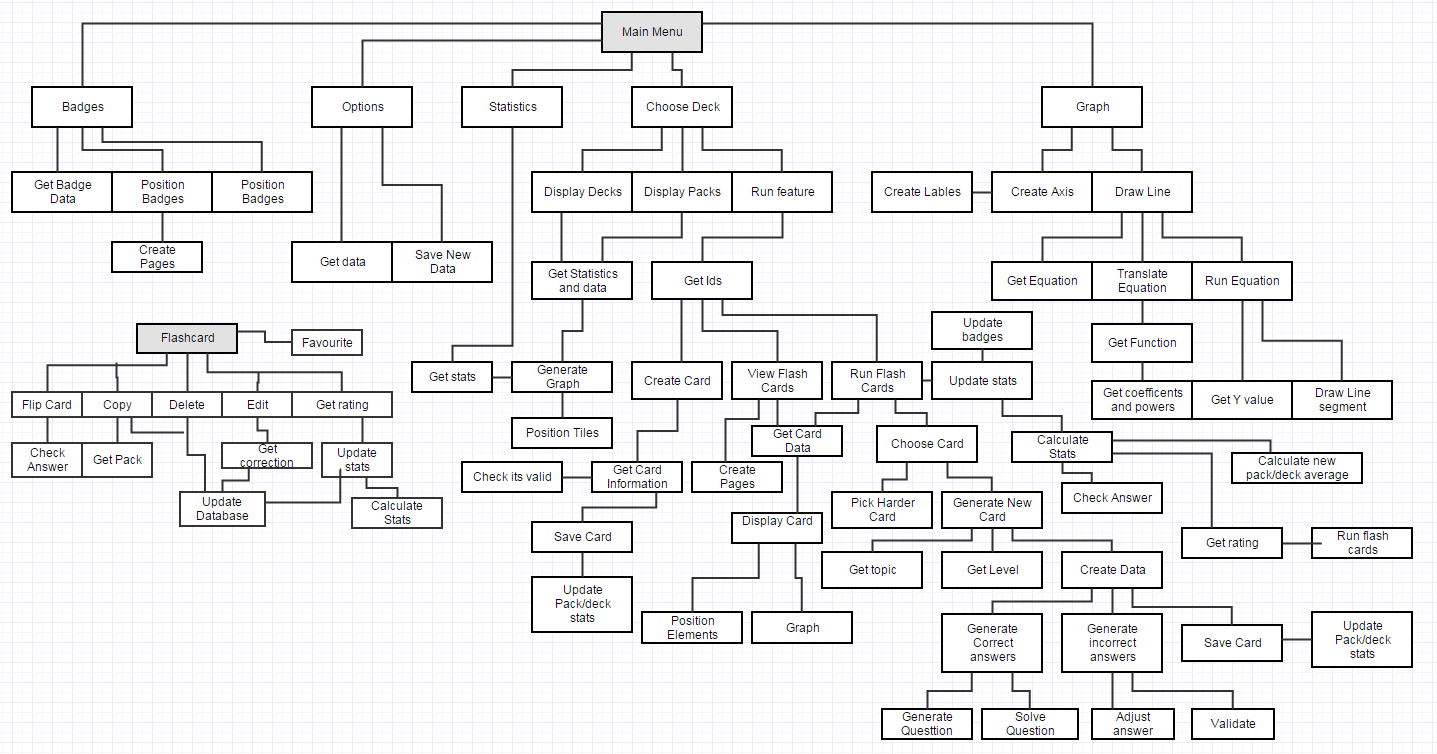
Design

Overall  system design

I wrote my program in Delphi , using Pascal as my programming language, to display the UI , perform the processes and save my data to a database. The database software I used was Microsoft access which stored all the data about my flashcards and statistics, this is accessed solely through dynamic sql within the delphi program.  
A level 1 data flow diagram can be seen in my analysis, which was made at the start however my program still performs the same processes so it's still got the same level one design.

|  |  |  |  |
| --- | --- | --- | --- |
| Inputs | Processes | Storage | Outputs |
| * New Flashcard Data * Load Existing flashcard data. * Users Name * If the User Wants an answer * Deck/Pack Choice * Editing Flashcard * User ratings * Graph Equation | * Add Card * Edit Card * Copy Card * Delete Card * Open Existing Card * Generate New Card * Get Pack * Create Graph * Create Stats graph * Get stats from DB * Calculate % completion | Database tables:   * Achievements * Badges * Deck * PackDeck * Pack * CardPack * Card   And Static Images that are loaded. | * Graph of equation * Flashcard with correct data in correct form * Statistics Graph * Decks, and Packs * Statistics for cards, packs, decks, and user . * Badges |

Description of modular structure of system

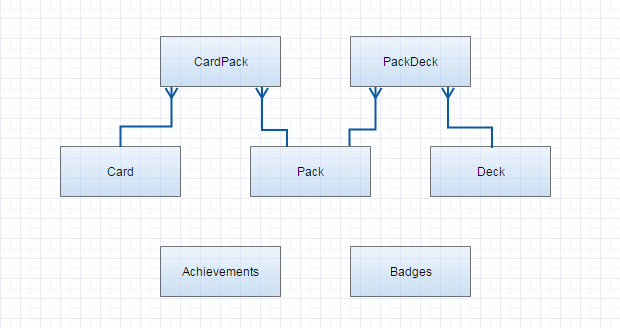


Above is a structure chart for the main processes in my program, it shows the processes of the program as a whole and the processes that can be performed on a flashcard as this appears in lots of places across the program.

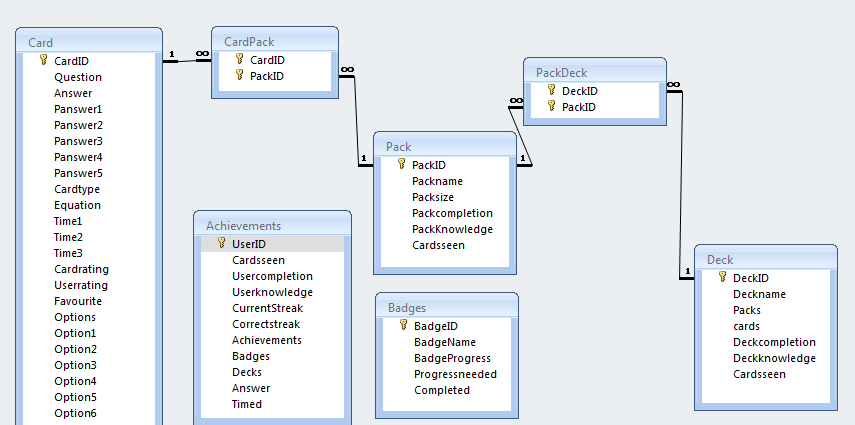
My systems UI uses a modular design with a static main menu at the top of the screen that can be used to change what is displayed upon the rest of the screen, this is explained later in the User interface design section.

Definition of data requirements

Below is the ER-Diagram of my finished database, which hasn’t changed since the analysis stage.

**

My database allows for one card to be within many packs, and the packs to contain lots of different cards. It also allows for one pack to be within many decks and one deck to contain many packs. This relationship allows the user to copy cards between packs, organize their cards at two levels, and copy packs into other decks.

Card(CardID,Question,Answer,panswer1, panswer2, panswer3, panswer4, panswer5,CardType,Equation,Time1,Time2,Time3,Cardrating,Userrating ,Favourite, Options, Option1, Option2, Option3, Option4, Option5, Option6)

Cardpack(Cardid,Packid)

Pack(PackID,Packname,Packsize,Packcompletion,Packknowledge, Cardsseen)

PackDeck(deckid,packid)

Deck(decked,deckname,packs,cards,deckcompletion,deckknowledge,cardsseen)

These tables are the five tables that store the cards and what pack, and deck they are in. They are set up so one card can be in many packs, and one packs can contain many cards, and the same for the relationship between pack and deck. It allows for this many to many relationship between packs/deck and cards/packs by having two tables showing how what each PackID is linked to which deck/card. This means that copying a card into a new pack only requires making a new entry in CardPack. This means that I have no redundant data within my database. My database also only stores atomic data within each of its tables, and has no partial key dependencies or non-key dependencies within each table.

Badges(BadgeID,Badgename,Badgeprogress,Progressneeded,Completed)

Achievements(UserID,CardsSeen,usercompletion,userknowledge,correctstreak,currentstreak,acievements,badges,decks,answer,timed)

These two tables done have a relationship with the other five as they are tables that store information that is only used by the program, and remains a static amount. However these two tables also contain no redundancies or non-atomic data, so are normalised to 3NF. The tables have been designed in a a way that if my user had requested it I could have allowed for multiple users to use the program and store the information separately using similar relationships between deck/achievements and badges/achievements, similar to the relations ships in the first five tables.

### Data design Dictionary

Here is an updated data dictionary, from the version seen in the analysis.

Below is a list of data that's being stored in the database, which will be located on the local hard drive (or equivalent) of the user's computer.

Note: Data Names in this tables have got spaces to make reading the table easier, the names stored in my database have no space. eg Card ID =CardID.

|  |  |  |  |
| --- | --- | --- | --- |
| Data Name | Data Type | Example | Validation |
| Answer. (in Card) | String | “x^2-2x+7” | Shorter than 255 characters |
| Previous answers. (1-5) | String | “x-2” | Shorter than 255 characters |
| Card Type | Char | “G” | 'o' , 't' , 'g', or 'f' |
| Question | String | “cos(2x)=” | Shorter than 255 characters |
| Equation | String | “sin(x+2)” | Shorter than 255 characters |
| Time | Real | 21 |  |
| Card Rating | Short Integer | 87 | Between 0-100 |
| User Rating | Short Integer | 3 | Between 0-5 |
| Card ID | Long integer | 12 |  |
| Deck ID | Long integer | 2 |  |
| Pack Size | Integer | 55 |  |
| Pack ID | Long integer | 22 |  |
| Pack name | String | “Identities” | Shorter than 100 characters |
| Deck name | String | “A2 Maths” | Shorter than 100 characters |
| Deck ID | Long Integer | 11 |  |
| Pack completion | Short Integer | 87 | 0-5 |
| Deck Knowledge | Short Integer | 45 | 0-100 |
| Deck completion | Short Integer | 55 | 0-5 |
| Pack Knowledge | Short Integer | 7 | 0-100 |
| Cards seen | Integer | 567 |  |
| Packs | Integer | 11 | <=11 |
| Cards | Integer | 67 |  |
| Decks | Integer | 2 | <=5 |
| Current Streak | Integer | 12 |  |
| Achievement progress | Integer | 55 |  |
| Correct streak | Integer | 12 |  |
| User knowledge | Short Integer | 10 | 0-100 |
| Users name | String | 'Amy' | Shorter than 255 characters |
| User completion | Short Integer | 22 | 0-5 |
| Favourite | Boolean | True |  |
| Time (1-3) | Integer | 23 |  |
| Option (1-6) | String | '17X^2' | Shorter than 255 characters |
| Options | Short Integer | 2 | Between 0-6 |
| Answer (in achievements) | Boolean | true |  |
| Timed | Integer | 12 |  |
| Badge ID | Long Integer | 2 |  |
| Badge Name | String | 'See 100 cards' |  |
| Badge Progress | Integer | 22 |  |
| Progress needed | Integer | 100 |  |
| Completed | Boolean | false |  |

Below is a list of some data being stored as you use the program, stored in the heap/stack of the system. These variables are not the same as the variables that are stored within the database if they have the same name, they are the ones used within the program , in processing etc.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Data Name | Data Type | Example | Used in | Validation |
| Current | Char | 'F' | Mainprogram |  |
| Cardid | Integer | 1 | Loading Flashcards | <50000 as any higher is an invalid card |
| Pid | Integer | 23 | Deck Selector / delete |  |
| Did | Integer | 11 | Deck Selector / delete |  |
| Cardids | List of integer | 1,23,35,56 | Loading Flashcards | Check it's not 0, as this means no flashcards are available to see |
| LastIds | List of Integer | 1,23 | Generating next card |  |
| Packnames | List of string | 'Modulus' ,'Differentiation' | Randomly generated cards |  |
| Count | Integer | 12 | Timed game |  |
| Question | String | 'what is 2+3' | Flashcards | Shorter than 255 characters |
| Equation | String | 'X^2+2' | Flashcards, Graph | Shorter than 255  characters, doesn't contain a function such as c(..) as this breaks the graph. |
| Options | String | '22' | Flashcards | Shorter than 255 characters |
| nFlashcards | List of Tflashcard |  | Flashcards |  |
| bFlashcards | List of tFlashback |  | View flashcard |  |
| Correct | Char | 't' | Answer checking |  |
| Flips | Integer | 1 | Flipping card |  |
| Previous answers | String | '77' | Card % calculation | Check db is not null |
| Previous times | Integer | 12 | Card % calculation | Check db is not null |
| Eq | List of Tfunction |  | graph |  |
| Polylist | List of tpoly |  | Graph, creating questions |  |
| X | real | 6.2 | Graph - Run equation | Check <> 0 when dividing |

I've used lists throughout my program as this provided me an easy way to have as many of one data type stored at once, and without wasting memory such as in a static array. These lists meant I could show a variable amount of objects on the UI at once, or store a variable amount of card ids, and names for use in algorithms that decided what card was to be shown. They were also very useful in translating , and running the graphing program as it meant I could store the functions in the equation in the order they happened, and what they were applied to, which gave lots more flexibility in my equations.

## Data volumes

All the data in the database will be stored in a Microsoft access database file on the user's hard drive. The overall size of this file is worked out below;

A user is likely to have created 3 decks plus the randomly generated questions as my user does 3 subjects. Within each deck there's likely to be 100-500 cards split across multiple packs. As the number of decks, packs, and achievements is much less then number of cards these can be ignored when approximating the data size.

An average flashcard stored, will contain.

* Answer: string, of length 255
* Previous answers, of length 255. Up to 5 unique answers.
* Card Type: as a char.
* Question: string of length 255.
* Equation string with length 255.
* Current user rating: short integer.
* Last 3 times for completion. Integer
* Overall card rating. short integer
* CardID: long integer.
* PackID: long integer.
* Favourite: Boolean
* Options: short integer
* Options1-6: String of length 255

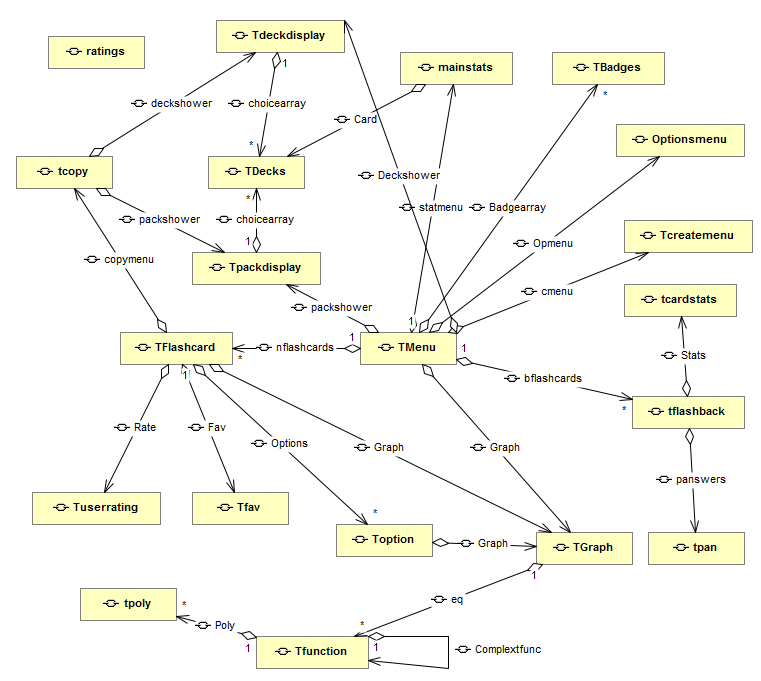
Overall size per card= 5000 bytes.

This means that in total the file will be 1.5-7.5MB.

My program also requires the .exe , a few pictures, and the user manual to be present which brings the total size to be 19-25MB. When being distributed a Microsoft access .exe is also included so the user will have the required software for the Delphi program to access the database, which brings the distributing total to 50MB (unzipped), or about 30Mb once compressed. This means that my program is perfect to be distributed online via a download link. This is because the program is small and quick to download, easy to send to other users if more people want to use the program, and there will be a backup of the program available online if they wish to use it.

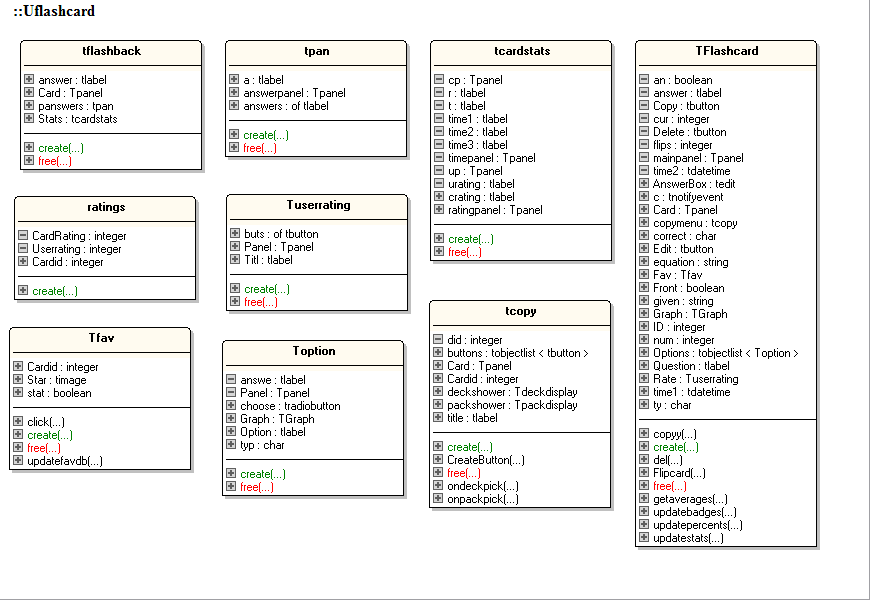
This method of storing and distributing my program is far easier for the user to use then it would be if I had given them a hard copy on a CD or Flash disk. As it enables them to install it where ever they need it without taking a storage medium with them, which could include giving it to friends.

While being run my program is going to be using at most two flashcards (10KB) , and could have up to a few hundred integers stored in lists as pointers , or ids which brings the running maximum memory usage to be no more than a few MB , which is fine for any modern day computer to handle.

Class definitions (diagrams) and details of object behaviors and methods

A Relationship Diagram for the objects in my program, the word on the line shows what that object is called in the containing object. Containment is shown in this diagram.

My program shows no inheritance between classes, which was originally planned however I decided to combine the three kinds of flashcard into the same class as the methods were all very similar, but not exactly the same for each meaning not much could be inherited. Also this made it easier to process the flashcards for use, and saved memory as less lists are needed to store the data.



|  |  |
| --- | --- |
| Class | Class description |
| Tflashback | The back display of a flashcard, without repeated buttons for use in viewing decks |
| Ratings | Used to store a cards ratings, when passed between functions, and used in determining next card to be show. |
| Tfav | The star icon that shows if the card is a favorite, and acts as a toggle. |
| Tpan | Shows previous answers on a tflashback |
| Tuserrating | Gives the user a way to choose a user rating |
| Toption | Displays an option for a flashcard, either with or without a graph. |
| Tcardstats | Shows card ratings, and times on a tflashback. |
| Tcopy | A copy window, with deck picker that lets you copy a card. |
| Tflashcard | The main Flashcard seen when running a deck. |

|  |  |  |  |
| --- | --- | --- | --- |
| Class | Function/Procedure Name | Description | Returns? |
| Tfav | Updatefavdb | Updates the cards favorite status in the DB | None. |
| Tcopy | Ondeckpick | Onclick procedure that's activated when they choose a deck. | Shows relevant packs |
| Tcopy | OnpackPick | Onclick procedure that's activated when they choose a pack | Returns screen back to flashcard. |
| Tflashcard | Copyy | Creates the copy menu | None |
| Tflashcard | Del | Deletes the card from the DB and updates pack numbers | None |
| Tflashcard | Flipcard | Flips the card to the other side if its entered correctly | Shows opposite card side |
| Tflashcard | Getaverages | Gets the average ratings for a pack | Average Ratings |
| Tflashcard | Updatebadges | Updates the badges in the database | Tells user if badge has been unlocked |
| Tflashcard | Updatepercentages | Calculates the new card rating | None |
| Tflashcard | updatestats | Updates all pack, deck, user, and card stats | None |

TFunction is designed with a list of Tfunctions within it to allow for an equation to have complex functions eg Sin(e^(x+2)+1) , the string that Sin is applied to would be classed as a complex function so when the object was created it would then be translated again into a list of functions for Sin to be applied to.

If the string was just a polynomial then no complex function would be added and the Tfunction would have a Polynomial list instead



|  |  |
| --- | --- |
| Class | Class description |
| TcreateMenu | Shows the ‘create a card’ menu, which is also used to edit a card. |
| Tbadges | The object that shows badge progress |
| TDeckdisplay | Shows the decks available, and lets you pick one. |
| Tpackdisplay | Shows the packs available and lets you pick some |
| Tdecks | One deck/packs statistics and name. |
| Tgraph | The graph that can be passed an equation to translate and draw |
| Tfunction | The format for storing one function, including what function , what its applied to , and its coefficient, as well as what function follows it in the equation, it has a Tfunction as an attribute so that I can have functions of functions of functions etc, and it will work out the y value correctly still. |
| Tpoly | Stores the coefficient and power for one polynomial in a series of polynomials. |

|  |  |  |  |
| --- | --- | --- | --- |
| Class | Function/Procedure Name | Description | Returns? |
| Tcreatemenu | Createbutton | Creates a button on the form and adds it to a list | None |
| Tcreatemenu | Createedit | Creates a edit box on the form and adds it to a list | None |
| Tcreatemenu | Createlabel | Creates a label on the form and adds it to a list | None |
| Tcreatemenu | Createform | Clears the menu, and calls the correct form to be created | None |
| Tcreatemenu | Oncreate | Checks the data is valid, Adds the card to the database, and calls createform | New form is created |
| Tcreatemenu | Onedit | Checks the data is valid, Edits the card in the database. | None |
| Tcreatemenu | Questionboxes | Creates the correct form for the option picked | None |
| Tdeckdisplay/  tpackdisplay | CreateOption | Creates the tdeck for the current pack/deck | None |
| Tdeckdisplay/  tpackdisplay | Furnish | Creates all the tdecks and correctly spaces them | None |
| Tdecks | CreateGraph | Gets the stats for all cards in the deck/pack and makes a bar chart. | Updates the graph picture |
| Tdecks | Del | Deletes the pack/deck from the DB, including all cards that are only in that deck/pack | None |
| Tgraph | Draw | Translates the equation to a Tfunction form, then gets a y value for every x value and draws this on the graph. | Drawn graph |
| Tgraph | Runequation | Gets a Y value for a given x value using the equation as a Tfuntion | Yvalue |
| TFunction | Stringtopoly | Converts a string into a list of Tpoly | List of Tpoly |

|  |  |
| --- | --- |
| Class | Class description |
| Mainstats | Creates and displays the overall stats for a user. |
| OptionsMenu | Creates and displays the options available to change in the program |

|  |  |  |  |
| --- | --- | --- | --- |
| Class | Function/Procedure Name | Description | Returns? |
| Mainstats | Createlabel | Creates a label on the form and adds it to a list | None |
| Mainstats | Createedit | Gets the users stats from the DB | Ratings, progress, cards, correctstreak, currentstreak, badges, timed. |
| Optionsmenu | Createbutton | Creates a button on the form and adds it to a list | None |
| Optionsmenu | Createedit | Creates a edit box on the form and adds it to a list | None |
| Optionsmenu | Createlabel | Creates a label on the form and adds it to a list | None |
| Optionsmenu | Onnamechange | Changes the name in the datatbase | None |
| Optionsmenu | Updateanswer | Changes weather an answer is required. | None |

Identification of processes and suitable algorithms for data transformation   
*Describe in outline the algorithms you have used.*

Calculating Y value for an equation given its x value ( procedure- Runequation in tgraph)

This Function takes a given x value and runs the translated series of functions on it to obtain a y value.

If current position in tFuntion is a polynomial then

Begin

Get ascii total of function in current position.

If function is a polynomial, Trig Function, Modulus, or Log then

X=Current Positions polynomial as a real

Result=function(x)

If function is n^x then

X=current positions polynomial as a real

Result=n^x

If function is E^x then

X=current positions polynomial as a real

Result=E^x

If function is (...)^n then

X=current positions polynomial as a real

For i=2 to length(function) do

Temp=temp+function[i]

N=string to real(temp)

If( x>0) or (x<0 and N is a integer) then

Result=X^n

Else reult=-1000000

If position=0 then

If polynomial isnt just a integer then

Result=result\*coefficent

Else if previous positions next operator was / or \* and polynomial isnt an integer then

Result=result\*coefficent

Else if its not an integer

Reault=result\*|coefficent|

else do

result=runequation(complexfunction,x,0)

Get ascii total of function in current position.

If function is a polynomial, Trig Function, Modulus, or Log then

Result=function(result)

If function is n^x then

Result=n^result

If function is E^x then

Result=E^result

If function is (...)^n then

For i=2 to length(function) do

Temp=temp+function[i]

N=string to real(temp)

If( result>0) or (result<0 and N is a integer) then

Result=X^result

Else reult=-1000000

If position=0 then

Result=result\*coefficent

Else if previous positions next operator was / or \* then

Result=result\*coefficent

Else

Reault=result\*|coefficent|

End

If position=0 then

If next=+, - or \* then

Result=result +,- or \* runequation(equation,X,Position+1)

If next=/ then

Temp=runequation(equation,X,pos+1)

If temp <>0 then result=result/temp else result=-100000

If posisition +1 next operation <> # then

If next=+, - or \* then

Result=result +,- or \* runequation(equation,X,Position+2)

If next=/ then

Temp=runequation(equation,X,pos+2)

If temp <>0 then result=result/temp else result=-10000

Else

If position-1.next<> / and position.next<># then

If next=+, - or \* then

Result=result +,- or \* runequation(equation,X,Position+1)

If next=/ then

Temp=runequation(equation,X,pos+1)

If temp <>0 then result=result/temp else result=-100000

End

The function looks at what the current position in the Tfuntions function is and sees if its applied to a regular polynomial or another Tfunction.

If it's a polynomial then run the function on the X value that comes out of the polynomial.

If the polynomial wasn't just a integer, or the previous function in the list was / or \* then result becomes result \* coefficient other wise result becomes result\* mod(coefficient).

If it was a complex function then run the equation on that Tfunction, and get a Value , use this Value to apply the function to and do the same as with a polynomial.

If it's not the last function in the list (next<>#) then apply the next function to the value from running the equation in the next position in the list. IF the operation was divide it skips over a function so only the correct parts are divided. Repeat until the end point (#) is reached.

Generating new flashcard, with a generated question ( procedure CreateCard in Uflashcard)

This procedure creates a card of the given topic at the correct level and adds it to the database and returns the cards id to the main program, This procedure makes use of a couple of other algorithms that I made;

* Differentiatepoly - Differentiates a Tpoly list into a new Tpoly list.
* Polytostring - Turns a Tpoly into a string with correct spacing, and operations.
* Getlevel - Gets the level that the next card is based off rating averages and cards seen.
* Tpoly.create - Creates a new Tpoly
* Generatepoly - Creates a Tpol list , of a variable length with variable coefficents.
* Differentiatefunc - Differentiates Trig , E and log and returns the result as a string.
* Generateequation - Generates an equation with any previous function , and differentiates it.
* Addcardtodb - Adds the card data into a new field in the database.

Answer=''

Question=''

Equation=''

Options=''

Randomize

If topic is Modulus then

Packid=2

Level=getlevel(1-2,averages)

If level =1 then

Polylist=nx^1+k (where the letters are random)

int=random between -3 and 3

A=integer to string of int

If int>0 then a= '+' + a

Coeficent=1 or -1

Equation='y=' coefficent + '|' + polytostring(polylist) + '|' + a

Question='which graph shows the line ' + equation

Op=random between 2 and 6

Typ='t'

Correctoption=random from 1 to op

For i= 1 to op do

If i=correct option then

Options[i]=equation

Else

Options[i]=random nX+k equation

Else if level=2 then

Do the same as level one but generate a polynomial of powers up to x^4

If topic=Differentiation then

Packid=4

Level=getlevel(3,averages)

If level=1 then

Questionpolynomial=random polynomial with 3 parts , with numbers in thirds

Question='Differentiate y=' polytostring(questionpolynomial) 'withrespect to x'

Answerpolynomial=differentiate questionpolynomial

Op=random between 1 and 6

If op=1 then

Typ='f'

Answer=polytostring(answerpolynomial)

Else

Typ='o'

Correctoption=random from 1 to op

For i= 1 to op do

If i=correct option then

Options[i]=polytostring(answerpolynomial)

Else

Options[i]=polytostring(random polynomal)

If level=2 then

Do the same as level one but generate a function of a polynomial with integer numbers and maximum two parts, the function can be Tan,Cos,Sin,Ln, and E^. Each of the functions are randomly chosen so they all can appear.

If level=3 then

Generate 2 functions of a nX+k form , and differentiate both.

Apply either the quotient rule or the product rule to get the overall answer

Generate options as in the previous questions

Add card to database

Get cards id

Translating a string into a Tfunction (procedure decodeequation in Ugraph)

Turns an equation as a string into a Tfunction that can be worked with, returns a Boolean showing if the equation is valid. This procedure requires the use of a few extra procedures that I wrote;

* Findstring - Returns the string that a function is being applied to, eg X^2+3
* Findcoef - Returns the coefficient in front of the function.
* GetNext - Finds the next operation after the current function : +,-,/ or \* , and where the functions finish position is.
* Tfunction.create - creates the function based off the details found, and makes sure its saved as a complex function if the string isn't a polynomial, the complex function is then decoded.
* Findpower - returns the real value that something has been raised to.
* Findinteger - turns a string eq , 2.2 or -3/4 or 45 into a real number, used in all the functions where a number is being found.
* Removex - Used to get the integer value that isn't connected to any function of x.

Result=true

Delete spaces in equation

Make equation uppercase

i=1

repeat

if equation[i]=S then

If the string is long enough for sin, sec then

If equation[i+1]=i and equation [i+2]=n then

Sect=Get string inside brackets

C=Get coefficient

N=Get next operation

Finish=Get finish point of function

Addfunction

I=finish

If equation[i+1]=E and equation [i+2]=n then

If it's not cosec then

Sect=Get string inside brackets

C=Get coefficient

N=Get next operation

Finish=Get finish point of function

Addfunction

I=finish

Else result=false

Else result=false

Else result=false

if equation[i]=C then

If the string is long enough for cos, cot then

If equation[i+1]=O and equation [i+2]=S and equation[i+3]<>E then

Get function details and add as before

I=finish

If equation[i+1]=O and equation [i+2]=T then

Get function details and add as before

I=finish

If its long enough for cosec and the letters match

Get function details and add as before

I=finish

Else result=false

Else result=false

If equation[i]=T

If the string is long enough for tan then

If equation[i+1]=A and equation [i+2]=N then

Get function details and add as before

I=finish

Else result=false

Else result=false

If equation[i]='|' then

Get function details and add as before

I=finish

If equation[i]=L then

If the string is long enough for ln then

If the string is long enough for log then

If equation[i+1]=O and equation [i+2]=G then

Get function details and add as before

I=finish

If equation[i+1]=n then

Get function details and add as before

I=finish

Else result=false

If equation[i+1]=n then

Get function details and add as before

I=finish

Else result=false

If equation[i]=E then

Get function details and add as before

I=finish

If equation[i]=( then

If i>1 then

Loop back to start and see if any functions are found if not then

Sect=Get string inside brackets

C=Get coefficient

N=Get next operation

P=Get power applied to bracket

Store function as '(Power)'

Finish=Get finish point of function

Addfunction

I=finish

Else

Sect=Get string inside brackets

C=Get coefficient

N=Get next operation

P=Get power applied to bracket

Store function as '(Power)'

Finish=Get finish point of function

Addfunction

I=finish

If equation[i]=X then

Loop forward and count brackets , then loop backwards and count

If it's in a bracket then

Check that before the bracket is n^k(..) if it is then

If base<>E then

Get function details and add as before

I=finish

Else do

Get function details and add as before

I=finish

If equation[i]=an integer then

Loop forward and backward and check that each way only has integers or . or / before a new operation if it does then, gets position of the integer

Get function details and add as before

I=finish

I=i+1

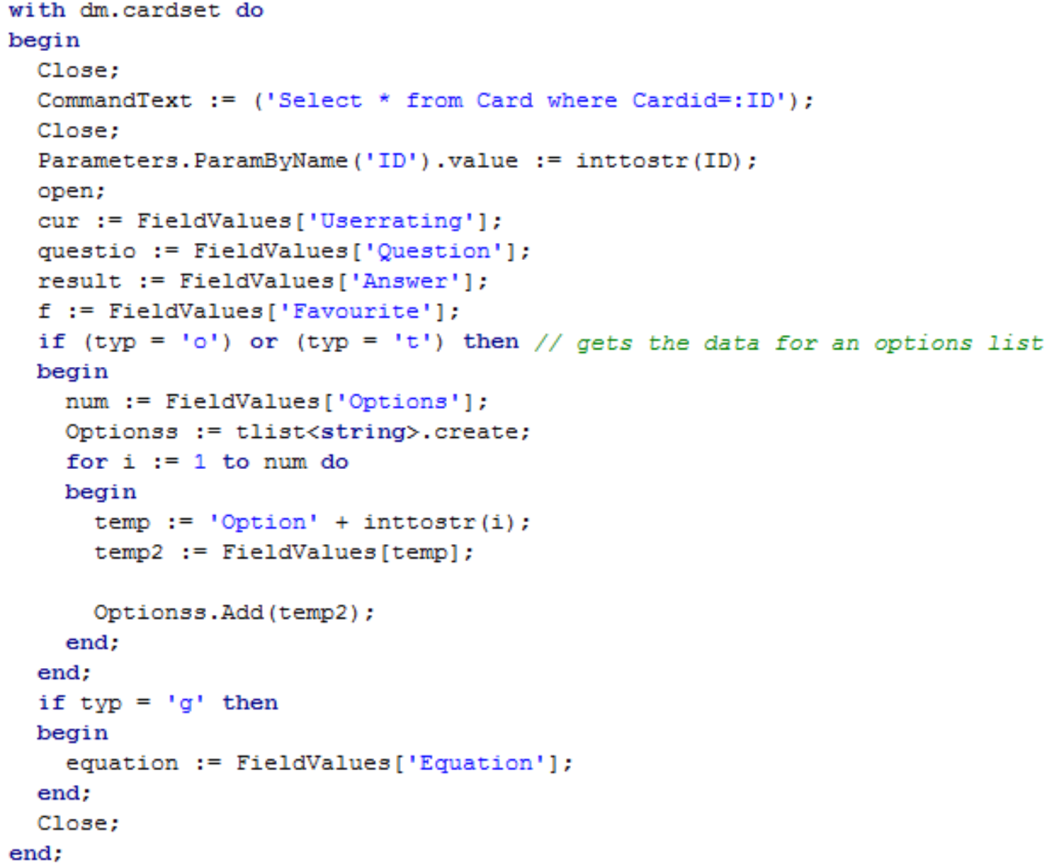
Until i=length of equation +1

#### **Sample Database queries**

Throughout my project I had to use dynamic sql to query, insert and delete data from my database here is a selection of some of the sql that I used.

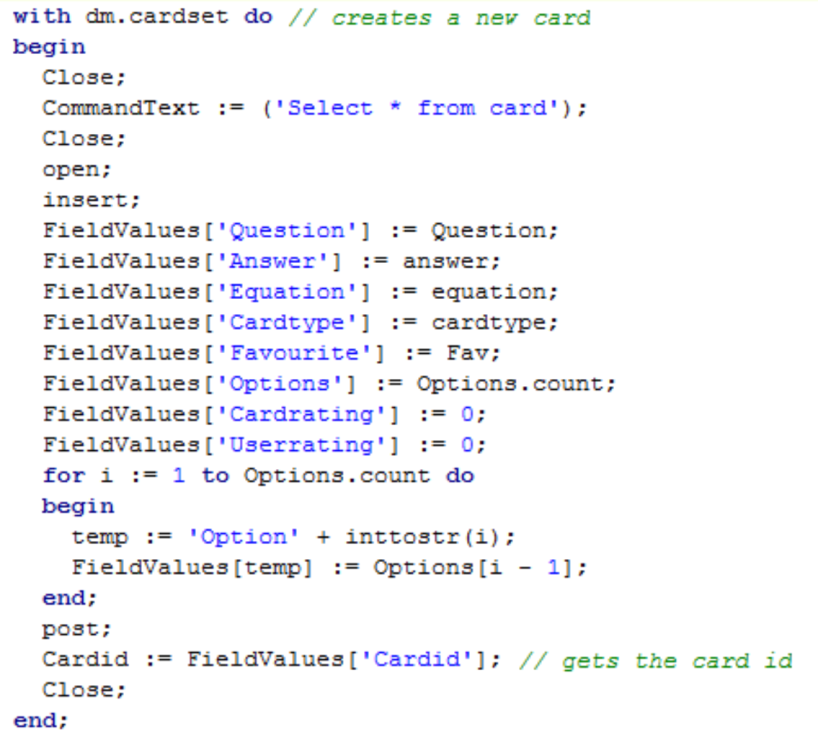
Getting card data

This sql is used whenever a card is displayed on the screen, its located within the oncreate procedure for tflashcard.



This sql is first loading all fields from the relevance cardid, and selecting the basic values that are used in all flashcards; userrating, question, answer, favourite. Then if it required the options and equation for the card type currently being made it will select these values as well.

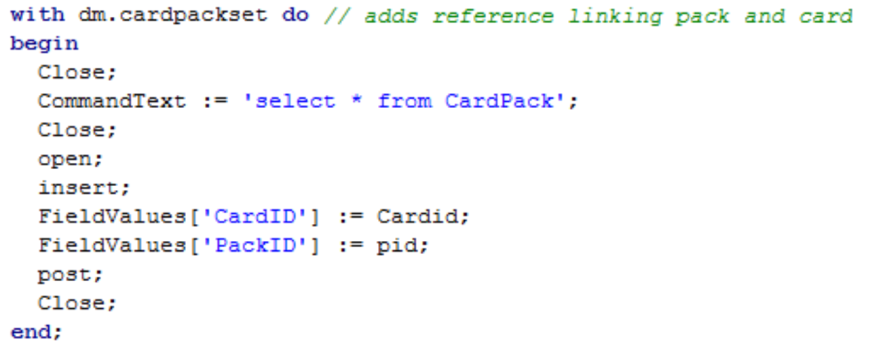
Creating a new card

The procedure ‘addcardtodb’ in Uflashcard handles adding new cards into the database, these cards come from the user or are randomly generated from the program. 

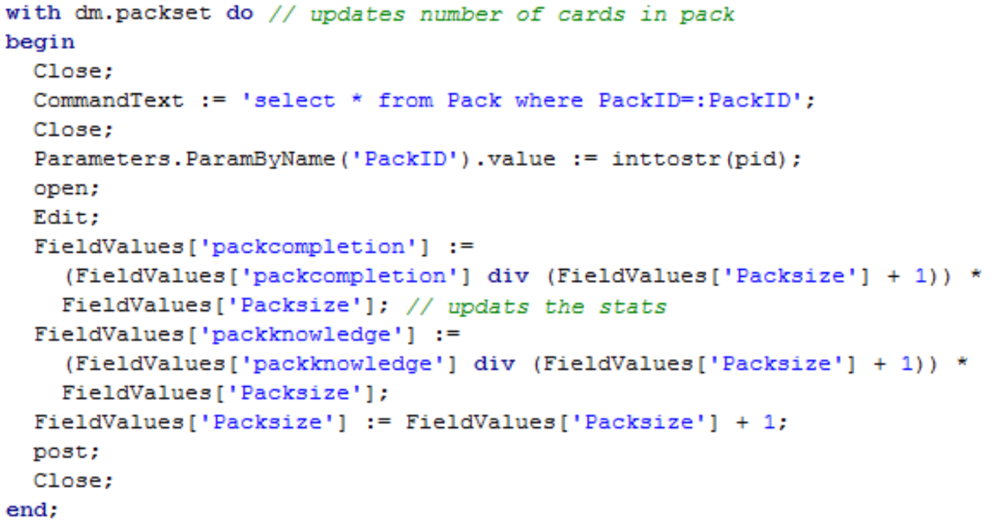
This sections is using sql to insert a new field into the table ‘card’ , the new field has the values that were specified when the card was created and defualt values for its stats. After its been created its cardid is then selected for use in randomly generated questions.

The sql statement for the code listed would be;

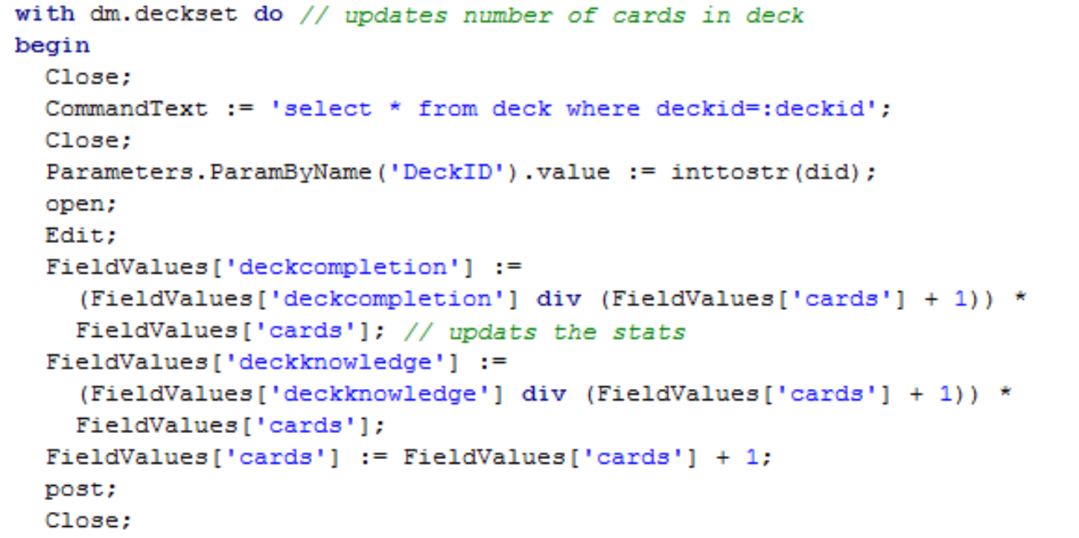
Insert into card (Question,Answer,Equation,Cardtype,Favourite,Options,Cardrating, Userrating,Options1, Options2,options3,options4,options5,options6) values(question,answer,equation,carttype,fav,options.count,0,0,options[0],options[1],options[2],options[3],options[4],options[5])



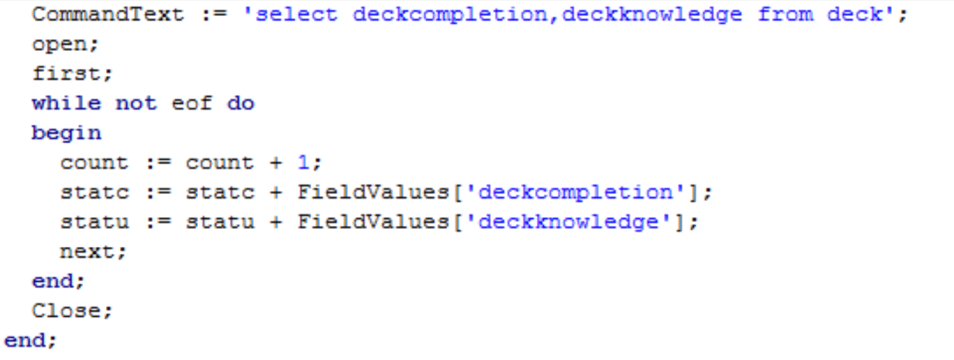
A new field in ‘CardPack’ is then inserted that links the newly created card to the pack it’s in.



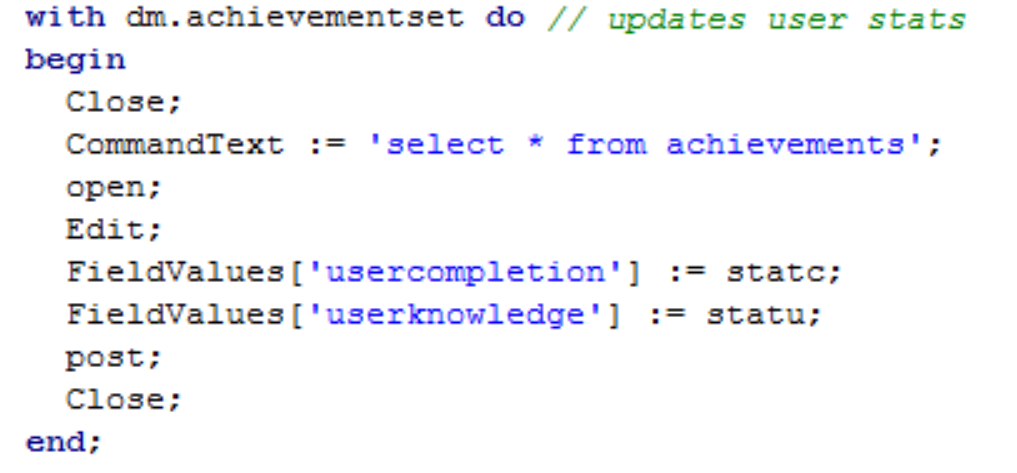
This sql selects the relevant pack by using its packid, it then increases the packsize by one, and decreases both the pack ratings by an amount proportional to the amount of cards currently in the pack. These three fields are then updated in the database.



This sql selects the relevant deck by using its deckid, it then increases the amount of cards by one, and decreases both the deck ratings by an amount proportional to the amount of cards currently in the currently. These three fields are then updated in the database.



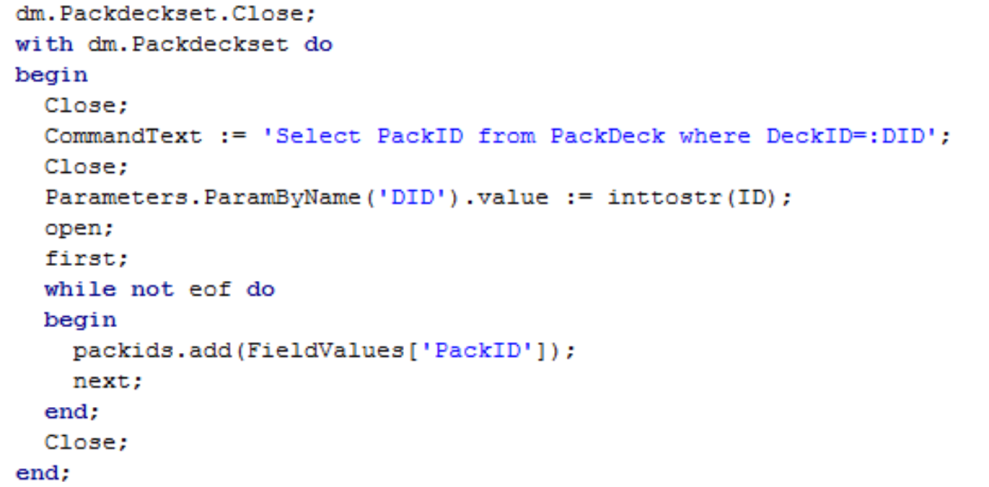
This sql is selecting all deckcompletion and deckknowledge decks within the database and adding up the ratings from both the sections. This is to be used to calculate the users average ratings for use in updating the achievements table.



This sql is updating the new average ratings in the achievements table.

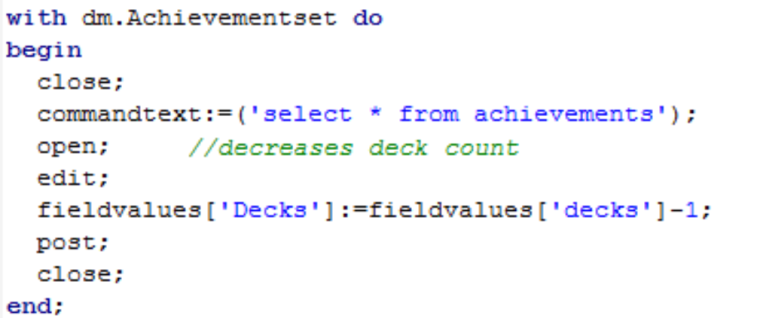
Deleting a deck

This SQL is called in the ‘del’ procedure of a Tdecks object when the ‘X’ button is clicked. It will delete the pack/deck from the database and any other packs connected to it, any non-duplicate cards will be deleted. And pack and deck number will be updated in their relevant tables.

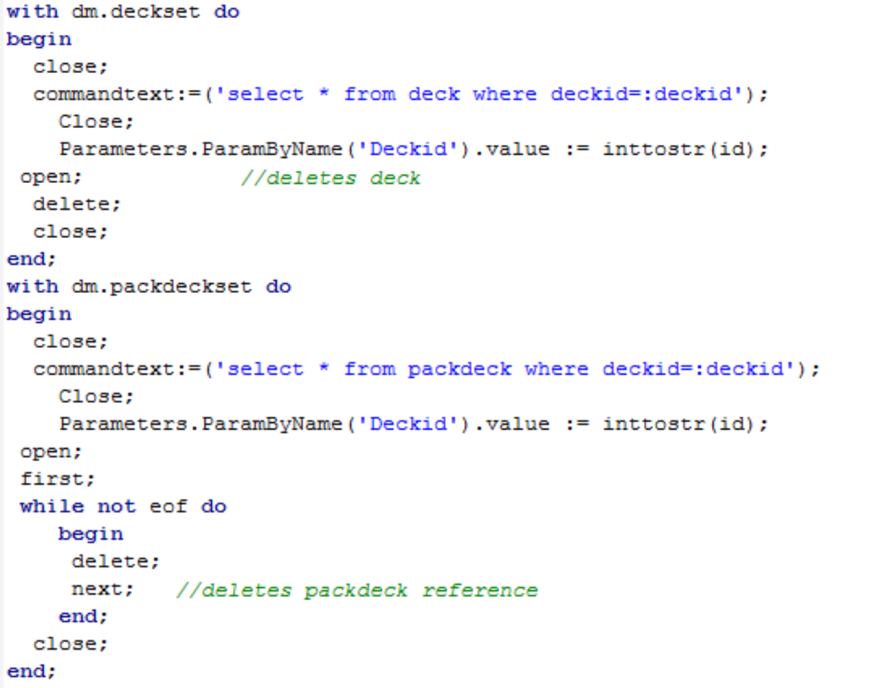


If a deck is being deleted then the first SQL used is to select all the packids that are linked to this decked as these will be deleted later.

Select PackID from packdeck where deckid=ID

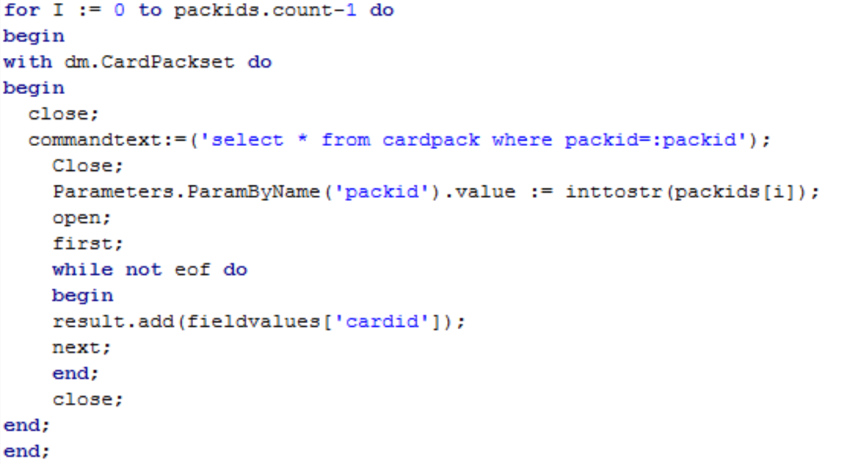


Deck count is then decreased in the table ‘Achievements’ using SQL to update the table.

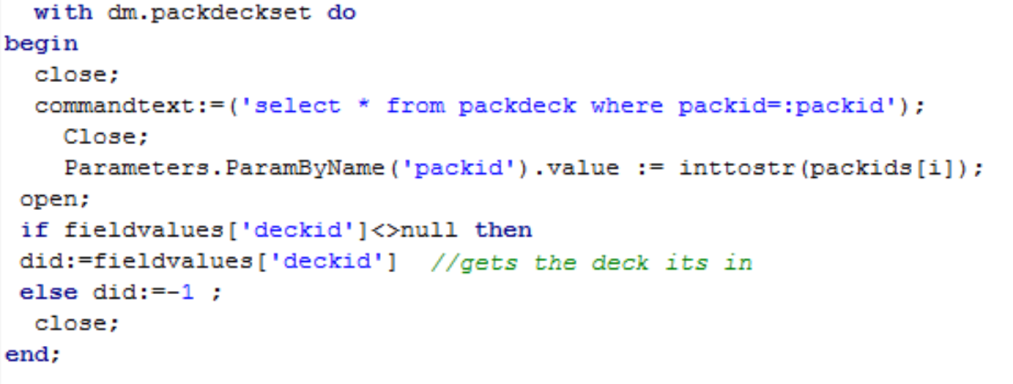


The deck being deleted field in ‘Deck’ is then removed and any references linking packs and decks are deleted from ‘Packdeck’, this step should have been enforced by my database package however this makes sure the deletion happens.

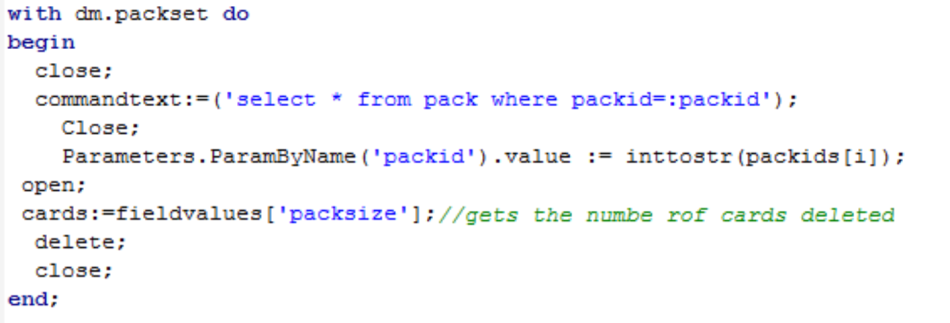
The next steps happen if either a pack or a deck is deleted.



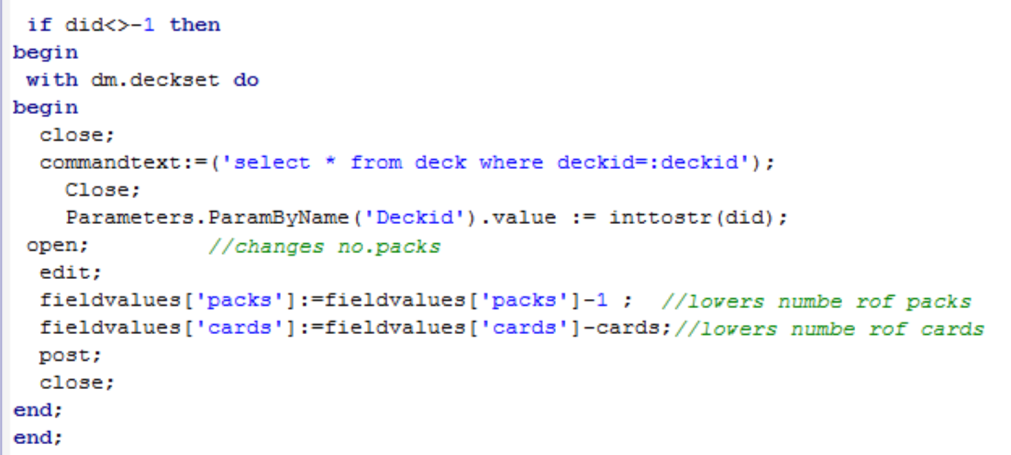
This SQL gets every cardid that is linked to all the packs that will be deleted. It does this by looping though the packid and selecting all fields from cardpack where the packid matches the current packs id.

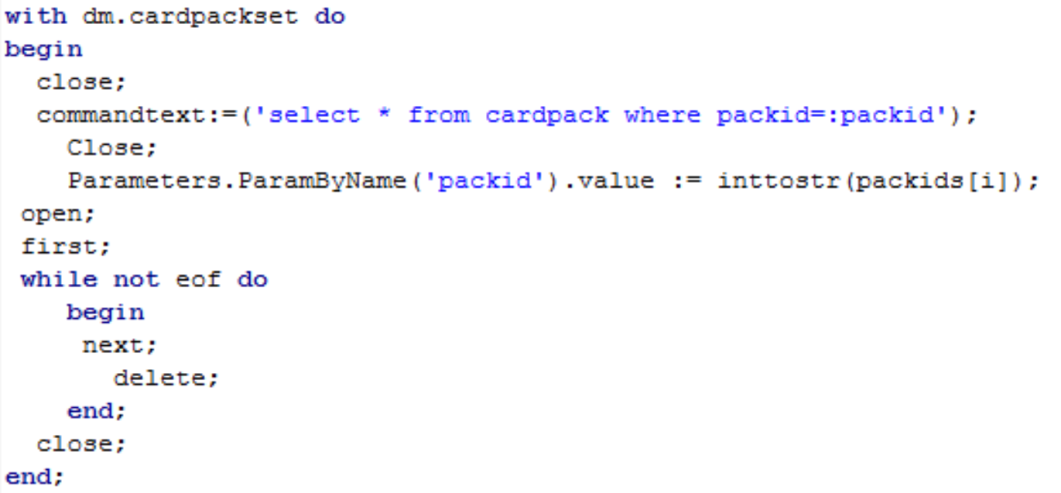


This removes any reference bewteen pack and deck that would still exist if only deck was being deleted. The SQL query is checked to not be empty to ensure it doesn’t try and delete an already deleted field.

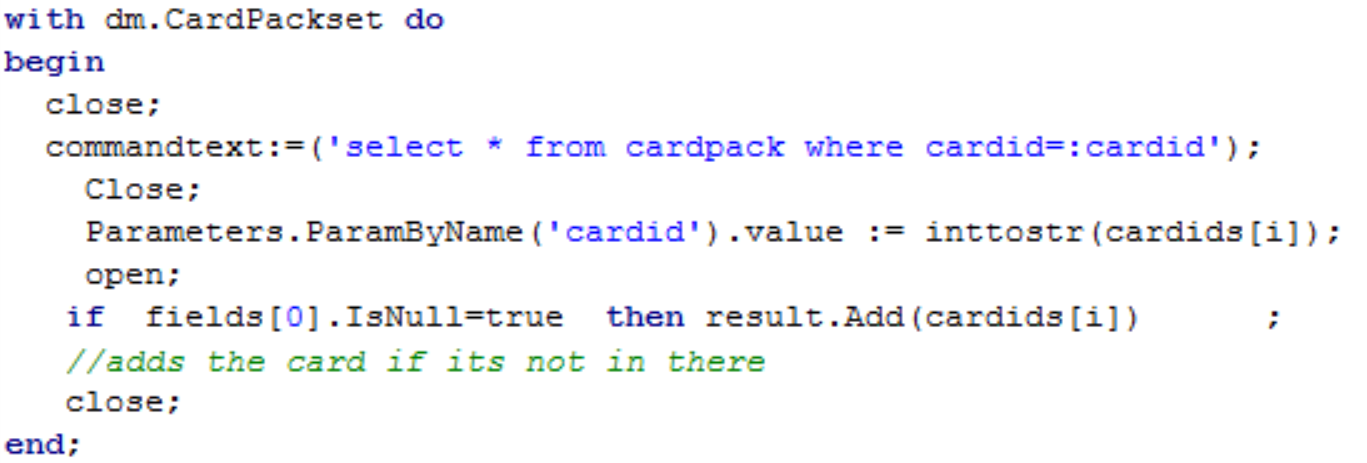


This SQL selects the number of cards contained within the pack being deleted by using its packid in the SQL, as if the deck containing the pack still exists its card count will need to be amended. After this number has been selected the packs field is then deleted.

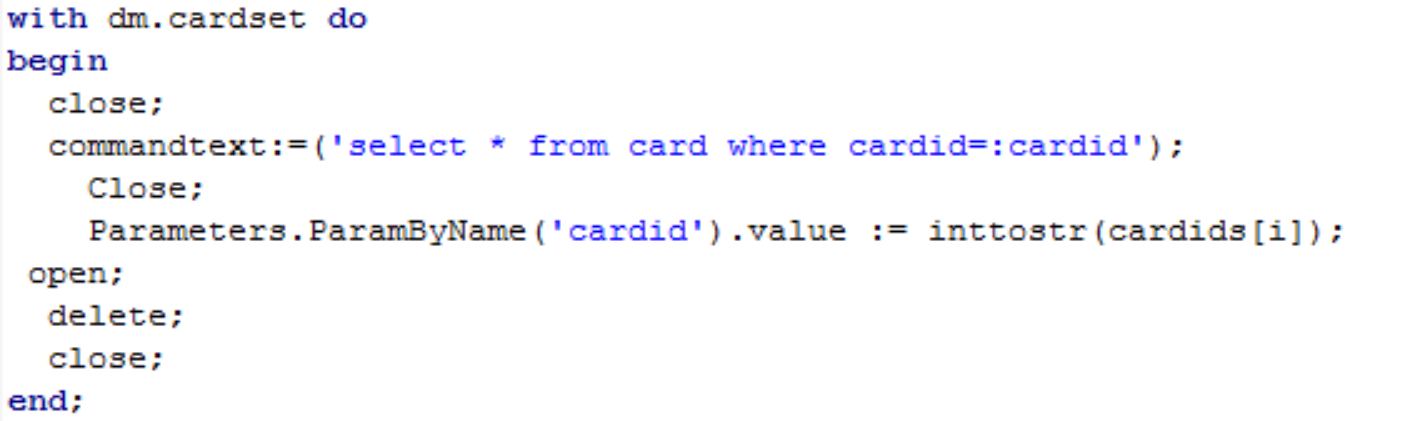


Next if only a pack is being deleted the pack count and card count is updated for the relevant deck field in the deck table by using its deck id when selecting it using SQL. 

The reference linking packids to cardids is then deleted from the cardpack table by selecting the fields linked with the current packid . This should have automatically happened when the pack got deleted but this ensures it does happen.



This SQL gets all the references for each cardid that still exists in CardPack, it ensures that the card doesn’t also exist in another pack and therefore won’t need to be deleted. The cardids to be deleted is adjusted based on the references found.



This is the final SQL required to delete a pack, this section deletes the cardids field from Card.

Other Uses of SQL

* Getting number of pack/decks for use in the deck shower.
* Seeing if an answer is needed for the questions.
* Updating favourite.
* Getting Badge progress.
* Updating and selecting the card and user rating
* Updating badges.
* Updating and selecting the statistics.
* Editing a card.
* Deleting a card.
* Copying a card.

User Interface design (HCI)

On the next two pages are prototypes of my user interface's design. These have been adjusted slightly in certain areas to make them easier and more intuitive to use.

The first two diagrams show the menu screens for the two biggest data capture methods in my program; Creating a card (and editing it) , and selecting a deck to use.

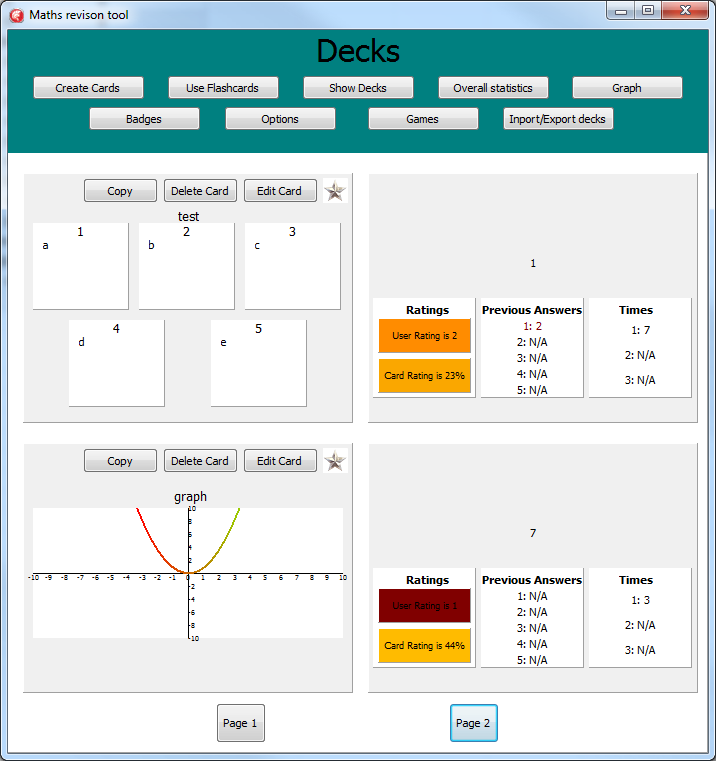
The next diagram shows the prototype for how navigation will work between each menu screen in the program.

These designs changes slightly in the finished program:

Shows if the card is a favourite, is gold if it is. Can be clicked to change state.

Title Showing what part of the program you're in.

Click this to bring up the edit card menu.



The Question

The Graph for a graph card.

Options on a options card

Clicking this opens up the deck picker menu, for choosing where to copy the card into.

Click this to delete the card, and recreate viewing screen.

The correct Answer

Shows the previous times for running the card

Shows the previous answers. They're red if it was wrong, and green if it was right.

Displays the card rating (0-100%) , and is coloured based on the rating, in a red-green spectrum. Card rating is determined by; Times, user rating, and correct answers.

Displays the user rating (0-5) , and is coloured based on the rating.

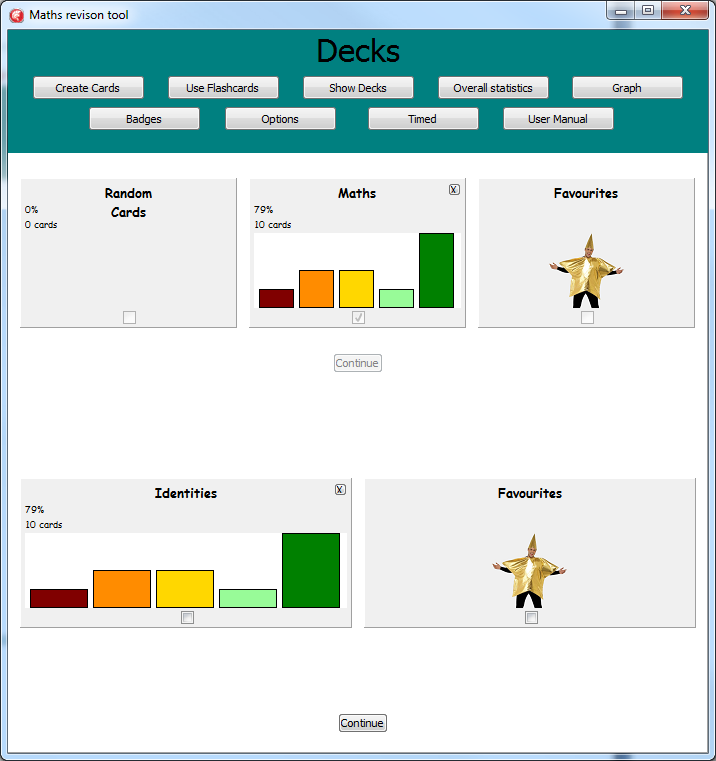
Page Navigator , each page shows up to two flashcards

The Picture above explains what each aspect of this menu does.

Above is what the deck viewer screen changed to. It's now shown in the same way that multiple pages of badges would be should, this ensures my program has a similar functionality across its different sections.

It also allowed for bigger cards to be shown , which made the information clearer and means more card statistics are able to be show.

The static main menu screen is in all my designs as it made it very easy to navigate to where you wanted to be , on any page you happened to be on.

****

Checkboxes to easily allow for many options to be selected at once.

Graph to visually show statistics

Pack/deck statistics

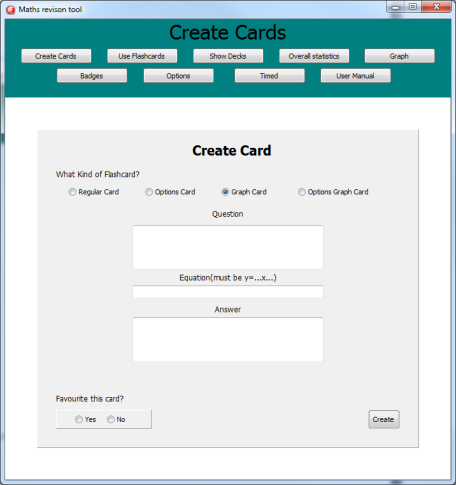
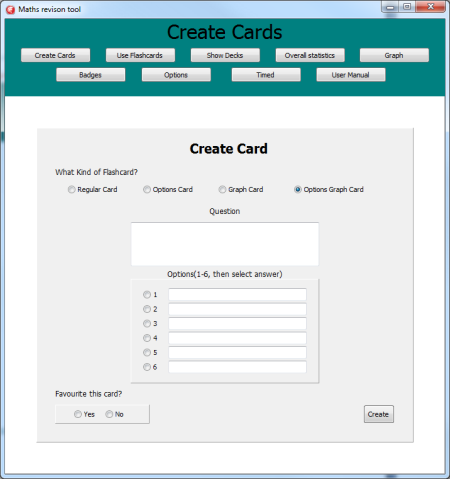
Available Packs, within the selected deck

Available Decks

Favourites options, is new deck when a card is being created

Above is what the deck selector menu changed to. It's now more visually appealing, provides more information and is easier to use as every option is shown with a checkbox to allow you to select the packs you want easily and quickly. The original design shown before would have been easy to use, however the newer one I felt achieved it better and my user preferred it.

This also easily allowed for error detection when a pack was empty etc , as I could easily see what options were selected and see if it was a valid combination.

****

Edit box to easily add a question

Radio buttons for easy selection of the available card types, positioned at the top as they are step one.

Radio buttons for selecting the correct option a card, makes it so it can only be one and can't be inputted incorrectly.

Radio buttons for selecting favourite a card, makes it so it can only be one and can't be inputted incorrectly.

Adds the card to the database, it's at the bottom as it's the last step.

The above is the new way of creating and editing cards. I've changed it as the new design only shows the information needed to be inputted , instead of formatting the screen to look like a flashcard. This makes it easier to use. Changing the screen to look different for each mode, is also much easier for the user as it's done with a set of radio buttons.  
Description of the measures planned for security and integrity of data

I Have Validation in place to make sure flashcards aren't entered with no data, or data in an incorrect format. These checks can be seen in the testing section. As this has been done it ensures that a user created card won't cause the program to break. The adding and editing a card section also are set to only accept strings up to 255 characters which is the limit that my database can store, meaning errors are avoided if they try and save a string that's too long. In order to make sure that true/false and options are selected and inputted with the valid values I've given them in the form of radio buttons meaning the user doesn't input any string and it can't malfunction. This validation is also in place when users are navigating around the program and if they enter an incorrect trig function into a graph, which stops the graph proceeding to an error.

No encryption is needed as the data is purely moving between being processed in the program and stored in the data base on the same computer, meaning it's not at a security risk if it’s left unencrypted.

My database will have referential integrity as every time any change is made to the data in one section which causes an update in a certain table, the program will carry forth this update onto the other tables by changing numbers or deleting references when necessary, and my code does this as well to check its happened. . This ensures that my program continues to function as changes are made to the database.

Description of the measures planned for system security

As my program will solely function from a single computer with a single user, I didn't need to include any passwords or security systems. However I could easily implement a simple password check, from the database, on the main menu if my user had specified they wanted this feature.

If the program needs to be reinstalled or the database breaks, my software is being disturbed through a download link online so an empty backup of the required filed can be found there if a problem occurs on the user's computer.

Overall test strategy   
More detail on my testing strategy can be seen at the start of my testing section (in a few pages) , and throughout it.

As I wrote more procedures and modules of my program I would test each of them to ensure they produces the correct outputs for a given input. If they didn't I would then follow the code through with trace tables or mentally with the data that produces a incorrect result and see what the problem was, after fixing this I could then continue to black box test the section and see if the problem was resolved.

When I was testing each section I would make sure to check that valid data worked , boundary data that was both valid and invalid produced the desired output, and that invalid data didn't cause invalid results to appear instead of an error message.

I would also test modules/procedures with each other to see if when combined they caused something to fail, and I gradually built up how much of my program was being tested, until I had tested the whole program at once.

I then gave it to my user for her to provide a short acceptance test and see if she found any problems that I needed to fix, or if something wasn't designed the way she had hoped for.