

Clipcode Usability Guide

Chapter 1 Usability for Database Applications

Objectives

The objectives of this chapter are to examine:

- Special needs of users when handling data via a GUI
- Requirements for database front-ends
- Dialog box layout and use of certain controls
- Keyboard usage for heavy data entry scenarios
- Special issues regarding handling large data sets
- Different ways of naming database records
- The use of domains and string-to-id mappings

Introduction

In this chapter, we will look at the usability requirements for graphical user interfaces for database applications. We will observe from the user's perspective.

The amount of attention that has been applied to usability in word processing or spreadsheet applications unfortunately has not been applied to usability in database applications. Certain database-specific issues need to be carefully considered at the design stage. These can substantially increase the ease of use of database applications when handled correctly.

Usability should not be considered in isolation - issues such as performance, transactions, network connections and middleware (i.e. non-GUI) have to be considered together and a coherent solution for all their requirements developed.

Types of Users

During the requirements analysis stage of any database project (indeed, any project), one critical issue to be specified is the expected type of user and usage. Questions that should be asked include:

- How much training will they receive?
- Are they computer-literate or not?

- How many hours per day/week/month are they expected to use the system?
- In addition to the core users, is it expected that others might use it on a temporary, infrequent basis? The answer is almost invariable yes, even if not originally planned!
- What are they going to use the system for - data entry, information tracking, supervision, reporting or administration? Are there any special requirements attached to these?

Let's examine two examples. Firstly, consider data-entry people who are constantly using the same application (e.g. entering magazine subscription information) and need to key in thousands of text entries each day. The "typeability" of the application is paramount - the "graphical" in GUI is not that important as they will seldom use the mouse. These users will enter the data using the keyboard only.

Secondly, consider supervisors in a financial institution, who must, among many other tasks, carry out supervisory tasks in a database application (e.g. approve overdrawn account for special reasons). The graphical nature and ease of use of the database front-end is extremely important. These users will tend to use the mouse and might need much more assistance and guidance from the application, in the form of wizards, help text, tutorials, and examples.

Performance

There is a substantial difference in an application's performance when it is connected to a small database compared to when it is connected to a large one. During the design stages of an application, it is important that some estimate of the data volume is produced (possibly integrated into the E-R/CASE tools being used).

For example, a listbox in a dialog filled with seven items might function satisfactory. However, when it is filled with 2,000 items it will be very slow and be practically useless, as it is not possible to effectively scroll through that many items.

When retrieving large volumes of data from the database for display it is important that the user has the option of, or is forced to, filter it in some way. For example, imagine a user asks an application to display the contents of a table containing 10,000 entries. The application could first display a filter dialog automatically requesting the user to select some filtering criteria and only fill the list once some filter has been entered. If no filtering criteria is entered then maybe the application should only fill the first 200 entries - so the user get a feel for the data in the database and might help her make a subsequent filter.

It is important to test an application with the maximum data volume - this will show up any weaknesses and allow them to be fixed before delivery to the customer. Indeed, if the max database is available early on in the design and development, different options may be considered while it is still simple to change.

Response time is a very important performance measurement - often the requirements document will strictly specify this and it will have to be provided, for contractual reasons. Techniques for speeding up the response time will need to be considered.

Issues Concerning User Interface Controls

We will now look at a number of important concerns that must be addressed when creating user interfaces for database applications.

Selecting and Editing

A typical user interface consists of displaying rows from a database in a grid in the main window, letting the user select one row, and then displaying a dialog-box editor for this selected row.

In-situ editing, (editing the selected row directly in the grid), is rare because of a few reasons. It will usually require a different locking on the database. The necessary grid-based user interface control support might not always be available. The `SELECT` used to fill the grid will probably be a join and might not be suitable for editing. Finally, by having a separate dialog for editing it is clear to the user when she is viewing (in the main window) or editing (in the dialog).

Validation of Text Input

You have three options regarding when you may do the validation:

- Immediately, as the user types each character

When a user is meant to enter a certain range of characters (e.g. digits) into a text-box, it is easy to check each character as it is entered. For example, a text-box in a dialog box that represents a `NUMERIC` field in the database will only allow characters between '0' and '9' to be entered. Slightly more complex but still possible is when the text being entered must have a certain format. For example, imagine entering a date – `DD/MM/YY` or `MM/DD/YY`. The validation code would have to take into account the international system settings and as each character is entered what previous characters have been entered and is the entry up to the current character correct.

- After the user navigates away from the field which is currently being edited

Certain fields must have entries. An interesting example of this in Windows Explorer. Try renaming an existing file to have a blank name or a name that is that of another existing file. Explorer gives you a warning and does not let you finish the renaming action until you enter a non-blank unique name for the file.

In a database application, if the user is to enter a primary key, this field must be non-null and unique. Quite often subsequent editing should not allow the user to edit the primary key as it may have been used as a foreign key in another table.

- When the user exits the dialog box

This is generally where validation is performed.

It is vital that the context of the error is communicated to the user. For text input fields the context is the actual text box containing the error? The sooner the user is alerted to the error the easier it is to decide on the context. You must ask yourself when does the application know an error exists – it could be that the user has partially entered some value and has not completed the task.

Another issue to be considered is the saving of invalid/in-complete data in the database, which may be corrected/completed later. For example, most bank branches are identified on a check using a bank sorting code (identification number). There is usually a simple numerical algorithm (e.g. based on a modulus) used to ensure these are correct. One may not run the algorithm until all characters have been entered. Assume the sort code is six characters long. Assume the user entered four characters and clicks on “OK” to exit the dialog box. What should we do? One option would be to display an error message box, explaining the problem to the user and once the message box is dismissed, the user must correct the problem. However users do not like being “trapped” in a dialog – the developer must offer a way out. Therefore, it might be better to display the error message box – and give the user the option of knowingly saving in-complete information to the database and exiting the dialog. One assumes that the user would later complete the information, when available.

Pick-Lists

Pick-lists are when we use a combo-box or list-box control to enable the user to select a textual value that they understand and which is associated with a numerical value that the database understands.

For example, imagine a database with an ORDER table and a CUSTOMER table. The CUSTOMER table has a CustomerID field and a CustomerName field. The ORDER table has a foreign key CustomerID that maps into the CUSTOMER table. When a user is entering a new order on-screen in an Order dialog, they could have shown to them the list of customer names, and select one, and behind the scenes the CustomerID is actually written to the database.

With the exception of data-entry staff, most people find it much easier to select a name rather than entering a code. The best of both worlds could be combining them. One could display a code and a user-friendly name (“13 – Order Processed”) and the user may enter the code (13) or select the item from the list.

When doing a `SELECT` on the database one can specify a sorting order. When adding items to a list control in a dialog one can specify that it sorts the items alphabetically. It is usually must better to get the database engine to perform the sorting because it will be quicker. A common method when associating an integer ID with a string name is to display the name in the list control and set its item data to match the string item. However, if the list control sorts the entries this will not work. In this situation, you must get the database to do the sorting.

It is strongly not recommended to display more than one or two hundred items in a list control. The user will have great difficulty scrolling through that many entries.

One interesting idea from Microsoft Access is the use of auto-expanding list-boxes/combo-boxes. As the user enters each character in the list is filled with matching entries. The user will never see the entire list. As the user types more and more, the selection narrows, until one is finally chosen. This is good for performance reasons.

Certain fields may only hold one of a pre-defined range of values. A common example is in an order processing system, the orders table might have a state field, which could have one of the values from the domain list {New, Processed, Shipped, Paid}. Domain lists are fixed and will not change (except from one release of the software to the next). Hence, they are ideal for pick-list filling.

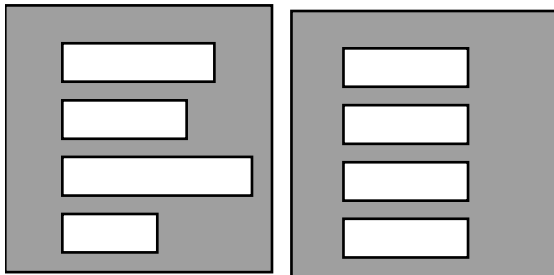
For performance reasons one could have it as a local database - but maybe it is not necessary - another option is to put it in with in the resources section of the application's executable, but then it is not available to reports etc. running in different executables.

Could use the server database and a local database for pick-list info etc. (Using Access, can have results of Server stored procedures output to a local database!). You can fill pick-lists quickly and repeatedly from the local database.

It is not just pick lists we're interested in - it is also list of entries in database (e.g. tracking messages). Let user enter filters and then execute retrieval function.

Layout

Should the length of text boxes reflect the size of the fields (which may be different, and hence un-aesthetic) or should they be aligned (thus conveying less information). I would recommend the first option, because conveying information is more important.



You have four GUI/database options when letting the user enter a multiple line text string, such as an address. In the user interface, you could let the user enter it in a single large text box, or you could use multiple text boxes. In the database, you could store the data either as a single field or as multiple fields.

If they are separate in the database then a reporting tool could let the user organize them in a variety of ways in the report, whereas if they are a contiguous block of text (with line-feeds) this is not possible.

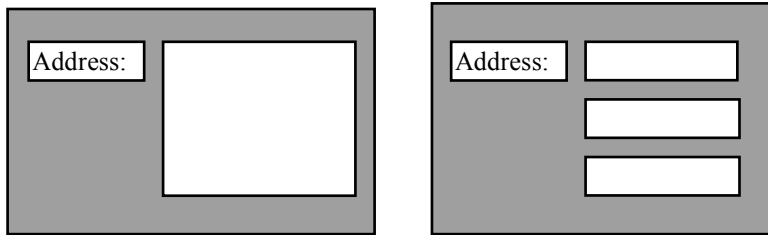


Table Editing using Property Pages

Novice and in-frequent users have difficulties understanding large dialogs containing many entries. A property sheet consists of a number of property pages and may be used as an alternative to large dialogs. The Options functionality under Microsoft Word's Tools menu is a good example.

For a database application, one could distribute the controls for fields in one table over a number of property pages, or where a group of tables are related display each table within its own property page and have all of them in the same property sheet.

Note that quite often the user will only see the first property page and will not bother to click on the tabs to display the hidden pages. Therefore, make sure that the fields that the user should edit are placed on the first property page.

Database Transactions in the User Interface

How are database transactions to be represented in the user interface? Some users know nothing about transactions, while others do.

Is the transaction to be opened as the dialog is displayed – and left open until the user clicks “Save/OK” (which results in the transaction being committed) or “Cancel” (which results in the transaction being rolled-back)? Alternatively, one could do a dirty read to populate the dialog box controls as the dialog box is being displayed, and then if the user clicks OK start a transaction, and write the values back to the table and then commit the transaction.

At the design stage of a database project, one checklist item should be to go through the entire design and consider transaction usage. One primary design goal is to have as few as possible transactions open, and when necessary to have them open for as short a period as possible. Remembering that the GUI might be only 5% of a database project - how the middleware performs is very important and it should not be blocked by transactions that are open for long periods. Therefore it is not recommended to leave transactions open while a dialog box is on screen – the user might take a long time to complete it (might even go for a cup of coffee while the dialog is on screen). The entire middleware processing should certainly not be stopped while this is occurring.

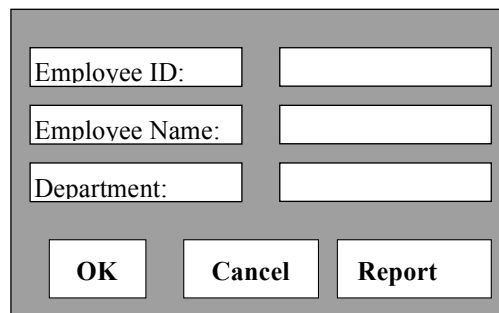
Fonts

Fonts serve two basic purposes in a GUI - to communicate from the user to the application, and from the application to the user - hence at least two fonts are needed, a system font and a user font. There should be a clear difference between the two fonts. The fonts should be consistent throughout an application.

Copy-ability of the Data Fields

Today's desktop is an amalgamation of a number of applications that are managed by the user to complete her day's work. Information must be shared among all these applications. Most users are now quite comfortable using drag and drop or copy and paste via the clipboard to transfer data between applications. Most document-based applications provided good support for inter-application data transfers (e.g. copying from a spreadsheet to a word-processor). It should be noted that in a document-based application, the document's data is displayed in the view, and data is transferred to and from the view. Dialog boxes are used to set options, etc., but usually no support is provided to transfer data between dialogs in different applications, except for single text strings (when a text string in a single edit box is selected, <CTRL>-C pressed, and the focus moved to a text box in a dialog in another application and <CTRL>-V pressed).

Up to now the integration of database applications with other applications has been poor. This is because typically the data from the database is displayed in a dialog box, and no easy method is provided of selecting a contiguous range (of all the field labels and the field data values). One could provide a "Copy" button that would collect up all the field identifiers and values and copy them to the clipboard, from where they could be copied into other applications. Another idea is to run a report directly from the dialog.



The image shows a dialog box with a gray border. It contains three rows of input fields. The first row has the label "Employee ID:" followed by an empty text box. The second row has the label "Employee Name:" followed by an empty text box. The third row has the label "Department:" followed by an empty text box. Below these fields are three buttons: "OK", "Cancel", and "Report".

Most database applications do provide a wide-range of ready-made or parameterized reports. It is possible to generate these reports and to copy from the reports to other applications. However, the reports tend to be totally separate from the dialogs, which means the user has to (a) select a report and (b) enter a large number of parameters. What one loses is direct manipulation. Ideally, what they would like is to be able to run the report from a specific one could provide a "Report" button in the important dialogs that would run the report that matches the data in the dialog.

Handling Complexity

The user interface in a database application may be the front-end to a very complex arrangement of relational database tables. The design goal is to hide complexity where possible but shows it where necessary. You should not limit functionality at the expense of "appearing" easy. This can end up making something much, much more complex as the user try to find away around the limitations of the user interface to complete some task.

Database applications tend to have a wide-range of users – power users who will know the application (and perhaps the database schema) in detail, normal users who will use the application most days and will gain a reasonable high-level understanding of it and beginners and occasional users who will need hand-holding as they perform their work. The range of knowledge they have about computer systems in general, databases, the application itself and the problem domain will vary considerably.

Quite often users are very intelligent; thought might not know that much about computers or the particular system in use.

Navigation

An essential point is handling complexity is to simplify navigation. Tasks should flow naturally from one to the other. It should be simple to navigate a hierarchy of data tables. The user may navigate using the keyboard, Windows, mouse or a component. The next logical step should be obvious (e.g. could disable buttons depending on data which has been entered).

If the user's hands are on the keyboard and task after next is also keyboard based, then it can be useful if next task is activated from the keyboard.

Usability Issues

Here follows a collection of ideas for handling database user interfaces. Depending on the application they might or might not be applicable.

The Dirty Bit

For applications that edit documents (e.g. a word-processor, spreadsheet, drawing package) it is quite common that an internal Boolean value is used to keep track of whether the document has been edited or not. This value is traditionally called the “dirty bit”. When the application is either closing a file or exiting, it checks if the dirty bit is set and after seeking confirmation from the user then writes the document's data to a file.

For database applications, it would be possible to maintain a “dirty bit” for a dialog that is used to edit a table. This would be of particular interest if the changes that the user could have performed in the dialog were time-consuming, and clicking on a `Cancel` button with no confirmation might involve loss of some work. By using a dirty bit, the application could seek confirmation from the user that changes are to be discarded.

Hiding the Concept of Tables

Many users of database applications are not aware of how the data is stored and designers should not assume that users understand the concepts of tables of relations among tables.

It is unfortunate that many simplistic database applications have a direct one-to-one mapping between a table and a dialog box. Often this will be the case, but on occasion it would be better to hide the concepts of tables entirely from the

user, and possible edit two tables in one dialog box, or a single table may be spread across multiple dialogs (or multiple property pages).

Shortnames

Most tables use an integer as a primary key. Typically, these integers mean nothing to the application user. If they have to select it from a list or enter it via a keyboard, there is no connection in the user's mind between the record that is being selected and how it is selected.

One solution to this problem is to use shortnames. These are additional fields within a table that contain a short character string (say 10 CHAR) and shortnames uniquely identify the record also.

Shortnames are aimed at end-users - and could be displayed in list-boxes, etc. and if used could avoid the need for applications to display primary keys at all.

Using Keys Outside the Database

Primary keys (and shortnames if used) uniquely identify a row in a database table. Often information relating to that row is also stored external to the database. This is particularly the case when bulky data, such as video, sound, large bitmaps, large EDI files, is stored on disk and "header" style info is stored in the database. The full pathname to the disk-based data could be stored as an additional field in the table, or using some scheme the primary key could be used to identify the disk-based information. For example, it could be decided that the images would be stored under a certain directory, and a file naming convention of <primary-key>.bmp is used to associate a certain disk-based image file with a particular row with that primary key in the database.

Add-On Editors

Most fields in database tables are edited on screen using simple user interface controls such as an edit-box, list-box or radio buttons. For more complex fields, such as a date, the application developer has a number of options. The easiest is to use a simple edit-box and try and get to user to use a specific format (mm-yy-dd, or dd-mm-yy or dd-mm-yyyy, or dd-mm-yy:hh:mm etc.). Another option is to add a simple calendar to the dialog. The problem with this approach is that it clutters a dialog with a numbers of controls (or for each day of the month) that have nothing to do with the table itself.

The third and best option is to use an add-on editor, which is a name associated with a secondary dialog (displayed after a user might click an edit button) and the add-on editor provides the necessary user interface to enter in a date and possibly time.

This concept could be used wherever users need to select complex data - such as a shortname from a particular table; or a registered user on a particular server, or a region of the world (based on a map).

Information Inheritance

Information may be defined at multiple levels in a database. The user interface for handling this scenario must display to the user whether the information is defined at a certain level, or “inherited” from a different level. In addition, the user interface for editing this information should be shared among all levels.

Regular Expressions

Using regular expressions to find entry quickly is strongly recommended. This can very easily be implemented by letting the user pass in wild-characters in searching dialogs. The application must pass these strings to the database using the SQL “LIKE” clause.

The Anti-GUI People

Now we will look at the concerns of data entry users. I have to admit that in an early draft this paragraph was titled “The Anti-GUI people” and it has gone through some renaming, (e.g. “User Interface for heavy-duty Data Entry”) but on reflection I think that many of their points are valid, so the title has returned to the original!

As we have seen, the data entry staff has different needs compared to other users. When entering the same data every hour speed and accuracy are the two main goals. Many data entry staff are paid on the number of subscription forms etc. they process. The “type-ability” of the database application is very important.

Graphical user interfaces for database applications have traditionally been designed for novice users to get them up and running. Less attention has been paid to users who use the database applications constantly. So, how may we help the data entry staff?

The entire application must be easily controllable via the keyboard. By this, we do not simply mean a few mnemonics, but rather careful consideration to all aspects of keyboard usage¹.

Imagine data entry staff entering information regarding subscription renewals. With their eyes on the card and their hands on the keyboard, they will read name, address etc. from the card and type it into the dialog box. When finished they will take a quick glance at the dialog box on-screen to confirm accuracy and then proceed to the next subscription card.

The number of keystrokes needed to complete any task should be minimized. Heavy data entry usually occurs when adding records. When editing records, staff must search through the database for the entry and edit it, so will probably be looking at the screen much more.

¹ The easiest way of doing this is to dis-connect the mouse from the developers’ machines for a few days and force them to use the keyboard.

Error Handling

An important issue in usability of database applications is error handling – are errors avoided where possible, is an alternative route to a solution quietly attempted and how elegantly are errors managed when they do occur.

Issues to be considered include:

- Rather obviously, preventing errors occurring is a good first step.

Developers should actively encourage users not to make mistakes. This includes making it clear the series of steps (and their order) necessary to complete a task. Dynamically changing the enabled state of GUI controls as appropriate. It is important to make it clear why items are disabled, and what actions the user may take to get them enabled again. For example, entering the primary key might enable the “Save Button” in a dialog. The fact that the primary key editbox must not be blank should be indicated, such as adding “Primary Key (Necessary Field)”.

- Back-Tracking

It is nice to be able to backtrack in the middle of a task. For example, a user who might have entered various parameters in a dialog box may have inadvertently clicked the dialog’s exit button. She should be asked whether she wants the data saved, not saved or simply return to editing the dialog box.

- Error Messages

Error messages should be clear, concise and accurate. They should indicate both the error that has occurred *and* how the user should proceed to fix it. Developers should not merely display a “System Error” message - its information content is very low!

Misleading errors are worse again. It is better to have no error message at all rather than one that does not state the facts and might send the user in the totally wrong direction to solve the problem.

For database applications, it is good practice to display all the error information returned by the ODBC driver (via `SQLException`). Note that the driver localizes the error message part of this information. Sometimes it is not very intelligible to the end user – an additional user message might also be necessary. However, if the user is reporting the error to the technical support department the `SQLException` information might be necessary. One option would be to display the application’s error message first and provide a “Detailed Description” behind a supplementary button, and in this supplementary dialog display the data from `SQLException`.

- False Information in the Database

Referential integrity should be used to prevent this occurring

- Domain Errors

Domain - specific entries are only meant to store values for the domain (OrderState). If for some reason the database stores a value that is not in the domain (a non-existent order state), then this must be handled. In theory, it should never happen, as domain values should be fixed. In practice, it usually happens during

development when a developer starts using a new state without updating the domains in the database first. If one does change the list of valid domains, one needs to decide what to do with existing rows that might use a domain value that was deleted or changed.

- Database Connections Errors

Sometimes the user will not be able to connect to the database. Other times during processing the database link will fail. Need to robustly handle these situations. For testing purposes need to stop database and start application, and alternatively have database functioning correctly and start app, and later stop the database (or simply disconnect from network) and ensure that application detects the problem - and rolls back etc.).

- Silent Error Handling

Would ensure that error reporting would be delayed temporarily and that a few retries would be made. Could be integrated into the database exception handler. (E.g. display “Please wait while accessing database” message if retry needed, and only if this fails then report error).

- Error Tracing

Need to find the cause of the problem at a programming level. The application could write detailed information to a trace file. The user could turn on ODBC tracing. It is sometimes beneficial to encourage the user to send this information to help-desk staff (via mail, print to printer, print to fax machine, save to disk).

Identifying Database Records

A number of different parties need to identify records in a database, either directly or indirectly. Many RAD tools emphasize the importance of the GUI operator, at the expense of all other parties, which is incorrect. Each entity has different requirements for identifying database records, which should be taken into consideration. Depending on the application, each entity’s importance is different.

Here is a list of parties and their requirements:

GUI Operator

Works on the client-side PC with a graphical user interface accessing the database. Could be involved in data entry, helpdesk or application administration. Based on various information, needs to enter/access records quickly.

External People

These people may never come near a computer - e.g. customer, student, and supplier. These will ring up a help desk with some identifying information and require action by the GUI Operator staff. The external people might provide unique identifying information (e.g. an identity number from a Student ID badge), shortnames, purchase orders numbers; or non-unique information (surname/first name, company name); or range information - “I sent the request last week” (e.g. filter may be set to “date field between Monday and Friday”).

Middleware components

these are usually the “meat” of a platform, and quite often are what really needs to speed up. They have no user interface and usually should deal with numbers solely. No use for shortnames. The use of numeric primary keys will speed these up.

DB administrator

Database administrators need to carry out backups, archival, emergency operations. They usually need not know about the contents of the database fields.

File System

Is the identifier going to be used for other purposes as well - for example, if files are to be associated with the entry (e.g. EDI files) these could be stored in a directory of the same name. (Note that an operating system numbers and characters are EXACTLY the same - no speed difference!). If middleware has the primary key, one might need an additional database lookup to get the shortname, to construct the path). If an ordinary user were to look at twenty 10-digit numbers that might be used as file names, the user has no idea which number is associated with which entry. If shortnames are used, then the user might be able to guess.

Mail Messages

Quite often mail messages are generated by/from the database and sent automatically to users and administrators or increasingly to remote (asynchronous) applications. This needs a user-friendly method of identifying records.

Reports

Often a report generator needs to create reports. These are normally to be read by people, so user-friendly names are required.

Types of Database access required by the different entities:

	<i>None</i>	<i>Read</i>	<i>Update</i>	<i>Insert</i>	<i>Delete</i>
GUI Operator		•	•	•	•
External People	•				
Middleware components		•	•	•	•
DB administrator		•			•
File System	•				
Mail Messages		•			
Reports		•			

Options for Identifying Records

Database records are identified using a unique *Primary Key* (which may consist of one or more fields). In addition, other fields or combination of fields (known as a secondary key) may be specified to be *UNIQUE*, and hence be used to access a database record, in the absence of the Primary Key. A shortname is a secondary key that is alphanumeric, and used to identify a record in a human-friendly manner.

Database records are accessed by a person or via other tables.

When via other tables one should normally use one or more long integers as the primary key.

Where users are involved, the choices are:

- Use a numeric Primary Key, visible to users
- Use an alphanumeric Primary Key, visible to users.
- Use a numeric Primary Key generated by system, with a unique Shortname generated by user, both visible to users
- Use a numeric Primary Key generated by system, with a unique Shortname generated by user, and only the shortname is visible to users

Primary keys may be generated automatically by the system or entered by user. In most cases, the user should assign shortnames and alphanumeric primary keys. Numeric primary keys should be assigned by the system (possibly based on some algorithm).

What the user enters, the user will definitely want to change. Hence numeric primary keys should be read-only, but alphanumeric primary keys and shortnames should be editable, possibly separate from editing other fields - e.g. via a “rename” button.

The Best Option

In many cases, the best option is to use a numeric Primary Key generated by system, with a unique Shortname generated by user, and only the shortname is visible to users. These combines the speed requires for middleware, with the user-friendly requirements for user interaction.

Use Numeric keys because:

- Alphanumeric names may be duplicates
- Name may be mis-spelt, for example when inquiring about a purchase order in a different country - much better have an invoice/customer number. Can be information rich - e.g. Our Ref = FR143 (143rd. purchase order from France) or Your Ref = BC-970343 (43rd purchase order from Bill Clinton, in March 1997).

- Numbers are sometimes quicker - letters sound different in different languages - but also are sometimes slower - there is no locality of reference with numbers but there is with names (if off by one letter it's O.K., but off by a digit - not O.K.).

Length of Numeric Primary Keys

In most cases primary keys should be 32-bit long integers. Where space is exceptionally tight they may be set to 16-bit, and for huge databases they may be set to 64-bit longs.

A 32-bit long will provide 4.2 billion keys which for almost all applications is appropriate. A 16-bit integer will offer 32 thousand keys and since keys are usually assigned on a max value, then this value can be reached surprisingly quickly.

A 32-bit long is a 10-digit number and most users would find it difficult to remember this - external people would need it printed somewhere.

Length of Alphanumeric Primary Keys

Alphanumeric keys ('0'-'9', 'a'-'z', 'A'-'Z') have a much bigger range and hence may be of smaller length to provide the same quantity of keys. They are slower to process (by middleware). They are much easier for humans to remember. (How many people could specify their Bank Sort Code - but everyone knows which part of town their bank is in).

Information Contained in Primary Keys

Negating that is that the main reason for using alphanumeric keys is that they may be based on some naming convention. (Student 97ENG045 started in 1997, is in engineering and is student no 45 in that year). These can make it easier for humans to remember them and to detect mistake (e.g. student 99ENG045 should not exist until 1999).

It should be evaluated whether information should be passed on in this fashion - for student numbers it is not a problem - but supposing the last three digits of the previous example was a customer number, in ascending order - the information that 45 items of a particular product were sold is vital commercial information and maybe should not be divulged. (If this is important random numbers could be used).

Numeric primary keys may be based on some sort of algorithm to detect mistakes - for example, the key modulus a specific number always equal some other - or a parity bit arrangement could be used.

CompuServe addressing "12345,6789" vs. Internet addressing "info@clipcode.com". On the other hand things can get out of hand and become too complicated - e.g. X.400 naming conventions - /c=ireland/admd=eirmail/o=clipcode/ou=eng/cn=info/

Domains

The permissible value that a column in a database table may hold is called its domain.

Types of Domains

For many columns of type CHAR or VARCHAR such as name or address there is no limited to the variety of values. Whatever the user types in is put in the column. Binary data (e.g. bitmap) certainly have no limited domain. Some numeric values (price, quantity and distance) also have no fixed domain (no fixed maximum).

Another type of domain (not really a domain at all) is when a column holds a foreign key into another table. This are fixed to the range of values in this other column in the other table.

Other columns do have a fixed minimum and maximum and the representation in the user interface, processing code and database is naturally the same. Age could be between 0 and 130 (is this the real max?), order date would be between the time customer account was set up and now, gender could be 'M' or 'F', percentage could be between 0 and 100. All these domains have a fixed minimum and maximum, and your software, during validation of inputs, can check that the values entered by users do fall within the required domain. All parts of your code can deal with this information in the same way.

Finally there is a type of domain has a fixed minimum and maximum, but the representation in the user interface, middleware objects and database itself are not naturally the same. They can be integer or char. It is this type of domain that requires special processing in your applications. We will see how in the rest of this chapter.

Multiple Representations of Domain Values

Take the example of an order processing application. Assume it has a table called Orders, and this table has a field called OrderState.

This OrderState field has a fixed domain - it must be one of the following values:

- Orders Received
- Being Processed
- Shipped
- Paid
- Cancelled

The user interface, the middleware objects and the database itself will need to deal with these values. How is it best to go about it?

From a user's perspective, the worst method is to allocate integers 1 to 5 to represent the states, and use these integer values everywhere.

From a database's perspective, the worst method is to use textual strings ("Orders Received", "Being Processed", "Shipped", "Paid", "Cancelled") to represent the above, and to store the strings in the database and in the middleware objects to do lots of (slow!) string compares.

The ideal situation is as follows. The user interface should display strings for the above values. The database should have integer values stored in it. The middleware objects will want to work with enumerated values for these.

Ways to Implement Integer/String Mapping

There are a number of ways in which the integers and strings for domains may be matched up.

Firstly one could store the strings in the combo-box or list-box directly, using the resource editor provided with your development tool. Then simply use the index of the combo-box selection to read and write the data to the database.

Use the index into a list. In the user interface, where you want to define the list of strings, add them to the listbox.

Another way is to define string resources as part of a string table. Each of these would be assigned a resource ID. The resource id would be exchanged with the database.

Another method is to use a configuration file containing the integer/string mapping. At program startup all the strings would be read from this file and added to the combo-box or list-box.

All these methods up to now will solve the GUI problem, but will not be useful when we want to use the strings outside the application, such as with a report generator or automated e-mail messages. Hence the best all-round solution is to use a domain table stored in the database, and each record contains the string and an identifier.

The Domain Table

Domain information is stored in a domain table that should be a static table in the database (i.e. not changed by the user - during run-time). One possible for a schema for a domain table consists of the following fields:

FieldName	Type	Comment
DomainID	Long	Identifies this domain group
LangID	char[3]	The identifier of the language used - taken from [Intl] section of win.ini (sLanguage)
DbValue	Long	The value stored in the database
StrValue	Var Char	The text string displayed on screen, in reports, mail messages etc.
BinValue	Var	Binary object, such as (small) bitmap/ WMF/ OLE object,

	Binary	displayed on screen, in reports, mail messages
--	--------	--

For 1ofN lists, the entry ID could be 1,2,3,4 etc.

For MofN lists, the entry ID should be bits from a long value, 1,2,4,8,16, etc.

The entry string is what should appear on screen and in reports - the entry ID is what is stored in the database. Another option is that the entry string is actually a resource ID - but this could cause problems for different programs accessing the same data (e.g. reports).

Note that if the binary object is language independent (e.g. does not contain text of speech) then the LangID field may be set to "STD", which will ensure that it is picked up for every language. This will be the normal case. The use of binary is not common up to now, but we believe it will increase in future - it may be used for reports and filtering.

Reasons why list indices should not be stored directly in the database:

- Items can be inserted (not necessarily at the end)
- Items can be dropped, thus affecting the list index of all other items

If the items are sorted alphabetically, then two users using different localized versions will have the list order different - using domains they will both be able to operate correctly simultaneously!

Handling complexity

Certain applications perform complex tasks. Do the user interfaces for such application have to be complex? Sometimes yes, and sometimes no. It is the role of the UI designer to make tasks seem as simple and as intuitive as possible, but in certain cases (some of) the true complexity must be evident.

By trying to simplify complex tasks for total beginners, UI designers can make an application completely un-usable for experienced people. This is a significant reason why different applications exist from the same software vendor covering the same area – to cater for different knowledge-levels of its users. For example, software for weather forecasting aimed at users with degrees in meteorology would be quite different from that aimed at enthusiasts.

User Interface for Server Applications

Users do not interact directly with server applications. When users need to invoke the services of a server application, they work directly with web browsers or desktop applications (also called clients) that in turn use some network protocol, such as HTTP, to communicate with a back-end server application. Often a server application provides services to other software systems, and no human user is involved.

When one talks about user interfaces for server applications it is in the sense of tools to help administrators. Server applications tend to be big, complex, and mission critical systems. It is a very serious problem within most organizations if server applications malfunction for any length of time. It is the role of administrators to keeping server them running correctly. These have the job of controlling the server application - starting / stopping it, performing customization, ensuring ongoing configuration, monitoring its behavior and tracking information as it flows through.

The administrator could be using the same machine that is running the server application, or the administrator could be using a different machine distributed somewhere on the Internet, or the administrator might not be a person at all – it could be another software application, such as a network management system, which wishes to manage many server applications, the network and the hardware in use.

On Windows, the Windows Management Instrumentation (WMI) APIs and runtime are provided to expose management information using COM. Administration front-ends, be they based on web interfaces or the Microsoft Management Console, can communicate with the WMI COM Objects to get and set management data.

The administrator will wish to start and stop the server application. Server applications are often implemented as Windows Services. Unlike a desktop application, these cannot be started simply by selecting an icon on the desktop. A service's running state can be changed using the OS-supplied Service Control Manager's MMC Snap-In (which can be accessed by selecting *Manage* from the context menu for the *My Computer* icon on the desktop in Windows 2000) or via the administrator's front-end for the particular server application, if available.

Usually starting and stopping a server is not instantaneous – during the interval between the command being issued from the administrator's front-end and the server completing its response to the command, the administrator's front-end should display suitable information to keep the administrator up to date with progress. Potential problem scenarios need to be expected and handled – for example, imagine multiple administrators are trying to change the state of the server application – some trying to stop it and some trying to start it. One solution here would be to "lock" the start/stop functionality, so that only one administrator at a time can be starting/stopping the same server application.

Or imagine an administrator trying to start a server application when the machine on which it is running is shutting down. (What happens if you start a desktop application and then quickly shut-down the machine?). The administrator should be informed and the machine should shut down.

Often server applications are dependent on crucial middleware components, such as a database engine. When these are not running, the server application can refuse to start, or start in a limited functionality mode, or itself start the middleware components. Which ever of these is decided upon, the user interface for the server application should keep the administrator informed.

Many server applications are composed of multiple processes. The administrator will want to keep an eye on these. Are they running? Are they processing messages as needed? How many threads and other resources are they consuming? Is anything blocking them un-necessarily? With groups of processes running inside a server application, there is usually some inter-process communication framework in place, whereby information can flow between the processes. The administrator will need a process monitor feature which presents the information collected from each process. As the process monitor might only be displayed once a week, it makes sense to collect the information for it only when the process monitor UI is displayed.

Server applications often rely on databases to store information. The information can usually be subdivided into that which is reasonably stable (customer lists, product descriptions) and that which is constantly changing, such as orders being processed. Entering and managing this information might be classed as administration on some platforms (e.g. e-mail/workflow scenario) whereas sometimes it could be classed as work for users (e.g. purchasing a PC online and tracking its manufacturing/delivery status).

For large server applications, there may be multiple administrators, and for security and organizational reasons, each may have different duties and security privileges. With a typical desktop application, the user is allowed use all commands. With the front-end for a server application, the user may be restricted in which commands he/she can activate. "Senior" administrators might be allowed start and stop the server application. "general" administrators might be allowed track messages and "information" administrators might be allowed edited stable data in the database (which users are permitted to send messages etc.).

The core functionality of most server applications is processing information flows. Terms such as messages / requests / orders / transactions are used to identify the discrete pieces of information that the server application handles. These are input somehow, the server app processes them in various ways, and results are sent out. The concept of a pipeline with multiple entry and exit points can be used to provide a high-level view of what happens inside many server applications.

There is often need to track messages as they flow through this pipeline. There can be errors with the format of messages. Attempted security violations needs to be detected. Proof may be needed that the message was actually received and processed (e.g. a financial payments system would need auditing information). When tracking information through a server application, one potential problem is being overwhelmed with information.

When a high volume of messages flows through a server application, vast quantities of tracking data can be collected. CPU cycles and hard-disk space are needed when collecting – so firstly some UI mechanism is needed to allow administrators to select a combination of the the type, status and volume of data to collect. Server applications can usually process multiple types of messages, and there is normally some type identifier inside the message. The status of messages could refer to those in error only, messages from a particular

source/for a particular destination, or all messages. The volume of data could be every message or messages for a short period at regular intervals (sampling). When tracking a particular message, administrators need to decide what information to store – the arrival / departure time of the message, its processing at each major step along the pipeline, or complete information. Administrators need a way of dynamically changing how much tracking information to record. It is not a once-off setting – it is an ongoing administrative task.

A simple mistake designers make is to allow the tracking database to fill up. This needs to be prevented, by archiving information at appropriate intervals.

Having asked the server application to collect information about messages, the administrator will probably sometime later wish to access the information for a particular message – but which one? "Management by exception" is often the motto. The administrator will usually be interested in just one message or a small subset of messages. Instead of having to study thousands or millions of messages, the administrator will those "exceptions" which he/she finds interested for a variety of reasons. Some filtering user interface is required for this. The final issue is how to display information about a message? It could be merely text output, but it would be particularly nice to graphically visualize the message as it flows through the pipeline.

Handling Large Data Sets

A user interface designed for a small number of data items can be hopelessly inadequate when faced with an unexpected substantially larger volume of data items. It is not simply a matter that the larger number will just slow down the application - it can often be rendered useless. What works in the small does not always work in the large.

For example, if a user has to select one entry from between five and twenty items, then a listbox would be the ideal way to presentation the options. If the actual number of entries grew to between one and two hundred items, then the user might grumble a little but a (sorted) listbox would still be manageable. However, if the number grew to ten thousand, it is clearly not at all useable. It would take too long to fill the list, it would consume too much memory, and it would take the poor user a long time to navigate through the list. Companies do make this mistake. A very easy version of one popular ISP's user environment did actually try to present ALL their registered email users in a list, so that instead of typing the recipient in the *To:* box, it could be selected with the mouse from a list. One could see how this feature was deemed very user friendly in the software development lab, with a dozen or two connected users, but in the real world with many thousands of users it was a disaster, and the product had to be quickly changed.

UI designers must try to estimate the number of data items for presentation to end-users, and also determine how accurate this estimate is likely to be. The number of days in the week is fixed at seven and (hopefully!) will not change. To select one of these seven radio-buttons would be suitable. The number of countries in the world changes slightly each year, due to various political / military maneuvers, but in general it is between 150 and 200, and is unlikely to fall outside that for a long time. To select one of these, a sorted listbox would be

suitable. The number of customers in an eCommerce database could range widely over time and no one can really predict the exact number. If we had to provide a customer representative with the ability to select information on one customer, we would first have to display some filtering/search capability, to enable the user to identify an interesting subset of all customers (e.g. those who live in a particular suburb of a particular city, and then this subset is displayed and the customer representative can select one. When displaying the subset, it should be in manageable chunks, say of fifty entries. This is similar to how popular web search engines, such as www.altavista.com, do it.

Faster hardware and a more expensive backend database will often (though certainly not always) improve data access performance. This is not the case with usability, as a variety of UI design decisions impact the volume of data a particular user interface can handle. If the data volume is going to be large but it is impossible to quantify, well that is fine – the UI designer can ensure that there is nothing which limits the amount of data that can be handled.

For types certain projects, data volumes often grow. There is a need to try to accurately estimate data volumes, and cautious UI designers multiply this by ten or a hundred, and design for that. It is extremely important that software engineers develop and test with a fully loaded database, as this eliminates the problem of delivering a wonderful application which is un-useable with realistic data volumes.