

Re: Beginning C# Q

Source:

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.framework/2005-10/msg00109.html>

- *From:* "Cody" <Cody@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx>
 - *Date:* Wed, 5 Oct 2005 12:01:09 -0700
-

Kevin,

Thanks again for the input. What is a typical method for handling multiple user inputs? It is very likely with the application I would like to create, that multiple users will be viewing and making changes to the same data sets. It seems like, when the user submits changes to the database that the users original dataset would be compared to the current dataset and verify to the user that changes had been made and have the user verify if they want to continue with the changes they are submitting. (I am not sure that is a very clear statement)

The data the user views is obviously just a snapshot of the database at the time the request is made to the database for a set of records. Probably more often than not the data is going to change in the database before the user submits their own changes to the database records.

Am I on the right track?

"Kevin Spencer" wrote:

- > Sure. However, be aware that a network application consists of a central
- > data store that is shared by all users, and multiple remote user interfaces
- > for connecting to and communicating with the database. If you're just
- > starting out with a network application, you have an awful lot to swallow.
- >
- > Designing your database is therefore, not quite the first step, particularly
- > if you're writing a network application. The database design is dependent
- > upon the requirements of your application. That is, you can't define a table
- > structure until you have defined, in detail, what data is going to go into
- > what tables. And that means defining your requirements.
- >
- > Assuming that your requirements are well-defined, as you say they are, you
- > can begin on the database structure. But one of your requirements is network
- > access to the data. This has to be figured into the mix. What happens when 2
- > or more users want to change the same data? Should the database include
- > row-level or table-level locking, which your client apps can then react to
- > when permission is not granted for a particular row or table? Then you need
- > to figure out the best table structure. How many tables will you need? What

Re: Beginning C# Q

> are their relationships, and how are they enforced by the database? For
> example, let's say you have a Users table, with multiple users in it, a
> Groups table, which defines which Groups users belong to, and a Permissions
> table, which defines permissions to certain types of data and operations. You
> will, of course, have one record per User in the Users table. But you may
> have many Users in a single Group. that Group can have multiple
> corresponding Permissions, and the Permissions table can have single
> Permissions granted to multiple Groups. Now, an Administrator wants to
> remove a Group. But there are users in that Group who may not belong to any
> other Group, and if you delete the Group, you are effectively removing ALL
> permissions for those users. So, you will need to define a 1-to-many
> relationship from the Users table to the Groups table, and enforce that
> relationship by preventing any Groups from being deleted if they have any
> User records attached to them. The Groups and Permissions tables have a
> many-to-many relationship, the most complex type of relationship, and you
> will probably need to define the rules that will prevent errors occurring if
> you delete a Group, or if you delete a Permission.
>
> So, you need to plan out the table structure and the relationship structure
> of the tables, based on your business requirements.
>
>> The application will not be performing very
>> many operations on the data in the database. The primary changes will
>> come
>> from the user directly.
>
> Not unless the user opens the database outside of your application. Your
> application is, and should be, the only "user" that does anything with the
> database. It will enforce the business rules. It is the intermediary between
> the users and the data. That is why an Interface is called an Interface. And
> keep in mind that the interface should ONLY do interface-related tasks.
> Separate the business process from the interface process. In the long run,
> you will be glad you did, or sorry you didn't!
>
> --
> HTH,
>
> Kevin Spencer
> Microsoft MVP
> ..Net Developer
> Big things are made up of
> lots of little things.
>
> "Cody" <Cody@xxxxxxxxxxxxxxxxxxxxxxxxxxxx> wrote in message
> news:FCDDDB15F-F379-4FE9-8C59-CA00D12CA1A2@xxxxxxxxxxxxxxxxxxxx
>> Kevin,
>>
>> Thanks again for the direction. I have a very thorough idea of what I
>> want
>> the application to do. What I really need is the best starting point.
>> The

Re: Beginning C# Q

>> project is database oriented. The application will not be performing very
>> many operations on the data in the database. The primary changes will
>> come
>> from the user directly.
>>
>> I chose Access originally because it is something I already know how to
>> use.
>> I am not against learning how to create a different database. I think I
>> can
>> figure out how to create the user Class. However, I was just starting at
>> the starting point of the application. That doesn't mean that is the best
>> place to start programming. In this case I would think that creating the
>> database would be the best place to start as it will ultimately be the
>> source
>> of all the data that the application will use.
>>
>> Since I am just learning it probably really doesn't matter where I start
>> with regard to my own perspective (just so long as I am learning and
>> making
>> progress).
>>
>> Thanks for your suggestions.
>>
>> Cody
>>
>> "Kevin Spencer" wrote:
>>
>>> Hi Cody,
>>>
>>> First of all, you're very welcome. I have had a lot of help along the
>>> way,
>>> and I like to give back at least as good as I've received.
>>>
>>> I knew you were a beginner, which is why I gave you the long explanation
>>> regarding the issues involved. One of the most difficult things to do
>>> when
>>> starting out as a programmer is to learn to think about the things we do
>>> without thinking about them (in other words, not to think like a user).
>>> One
>>> of the most difficult things to do after being a programmer for a long
>>> time
>>> is to think like a "normal" person. ;-) But of course, I wouldn't have it
>>> any other way.
>>>
>>> Computers are not nearly as smart as people think they are. In fact,
>>> they're
>>> quite stupid. They can't figure anything out for themselves, and must
>>> have
>>> explicit and accurate instructions about everything they must do. And
>>> they
>>> can't fill in the blanks, like humans do. Which is one reason they're so

Re: Beginning C# Q

>>> reliable compared with humans. After all, filling in a blank is often no
>>> more than an educated (or not) guess. Since computers are unable to
>>> guess,
>>> and therefore unable to make mistakes, they perform beautifully, always
>>> doing what they're told.
>>>
>>> But they have to be told every detail of what they should do. And to
>>> program, you have to instruct every detail to them. This forces one to
>>> become anal-retentive about logical details, especially after several
>>> years
>>> of suffering from one's own omissions.
>>>
>>> So, let's get down to your requirements. I still think you're starting at
>>> the wrong place here. The first requirement for any program is, what
>>> should
>>> it do? What job or jobs does it perform? My Uncle Chutney has a saying
>>> which
>>> I live by "Big things are made up of lots of little things." The best way
>>> to
>>> start on an application is with a small piece of it. And the first and
>>> most
>>> important piece is the functionality of the program. A program performs
>>> tasks. It can perform few or many. A Windows form is a user interface, a
>>> connection between the program and the program itself. It should never be
>>> confused WITH the program itself. It is a "universal translator," which
>>> speaks human at one end, and computer at the other. So, one of the first
>>> lessons to learn in programming is separation of interface from business
>>> logic. Your interface should perform no business logic. It should not
>>> perform any tasks at all, other than tasks related in presenting
>>> information
>>> to the user, and presenting information to the business layer.
>>>
>>> The business layer should be like the engine of a car. It does all the
>>> work,
>>> but is only connected to the user via the gas and brake pedals, the
>>> ignition
>>> key, steering wheel (actually, that's another system altogether), etc.
>>> The
>>> "engine" of an application is a class or classes which have no user
>>> interface, but only a programming interface (API). Like a car, it may
>>> actually have several systems or components that are linked together,
>>> like
>>> the engine, transmission, drive train, electrical system, etc, in a car.
>>>
>>> Now, a user login is certainly a component of the application. But it
>>> should
>>> be developed first as a separate class, or group of classes. For example,
>>> it
>>> might consist of a "User" class, which represents all of the stored and
>>> volatile properties related to the User him/her self. This would include,
>>> for example, the User's name, an ID, a Password, and any other pertinent

Re: Beginning C# Q

>>> information. This would be tied to your database. However, again, let's
>>> not
>>> mix functionality. You will need a database layer that talks to the
>>> database
>>> and delivers "raw" data to any class that needs it. It should not be
>>> dependent upon any class, but simply present an API for fetching, adding,
>>> updating, deleting data in the database. The "User" class can then get
>>> the
>>> necessary information from the database.
>>>
>>> A User Login generally consists of identification data being set in the
>>> User
>>> class, and the data compared against the corresponding data in the data
>>> source (database). If the necessary data matches, the login is
>>> successful;
>>> otherwise not. I hope you can see how this can all be managed without a
>>> user
>>> interface. For example, the UserID could be a settable property in the
>>> User
>>> class, and the Password another. They would be tied to a query that
>>> checks
>>> them out. Or, a more common method is to create a method that takes a
>>> UserID
>>> and password, and performs a query, returning "true" if successful, and
>>> "false" if not.
>>>
>>> Once built, this needs to be tested. This is where your form comes in.
>>> You
>>> would want to create a simple interface for inputting a user name and
>>> password, and calling the method in the User class.
>>>
>>> If this is a network app, Access isn't the best candidate to use. Access
>>> isn't really designed for network usage, although it can. And of course,
>>> if
>>> you want encryption, you will need to encrypt it at one end, and decrypt
>>> it
>>> at the other, to prevent data from being intercepted, for example, via a
>>> packet sniffer, and stolen.
>>>
>>> Test his part thoroughly before moving on to the next task. How do you
>>> eat
>>> an elephant? One byte at a time! ;-)
>>>
>>> I hope this gives you a good head start.
>>>
>>> —
>>> HTH,
>>>
>>> Kevin Spencer
>>> Microsoft MVP
>>> ..Net Developer

Re: Beginning C# Q

Re: Beginning C# Q

>>> Big things are made up of
>>> lots of little things.
>>>
>>>
>>>
>>>
>>> "Cody" <Cody@xxxxxxxxxxxxxxxxxxxxxxxxxxxx> wrote in message
>>> news:E5866CF3-2B11-4E76-BE53-1D58B372F5F2@xxxxxxxxxxxxxxxxxxxx
>>>> First of all. "WOW". That is a lot of good information. I should
>>>> qualify
>>>> the statement that I am a beginner. I have taken various courses on
>>>> programming. I can program fairly well inside EXCEL with VBA. When I
>>>> referred to a userform I just meant that I have used "Visual Studio.NET"
>>>> designer to create a windows form with the proper controls. I am
>>>> trying
>>>> to
>>>> design an application that will be used on a network. The first
>>>> control
>>>> will
>>>> actually contain the names of the allowable users on the system. I
>>>> would
>>>> like to store specific information for each user, as they will be
>>>> viewing
>>>> different information depending on their role in the company. I would
>>>> like
>>>> to use some form of encryption for the passwords.
>>>>
>>>> I am using an Access database for the system anyway. If it is
>>>> beneficial
>>>> to
>>>> store the users information in the database I have no problem going
>>>> that
>>>> route. I have read in various places that this information is often
>>>> written
>>>> to XML files and I don't know how to do that.
>>>>
>>>> Also since I have Visual Studio.NET I am open to a world of various
>>>> options.
>>>>
>>>> Thank you for your thorough reply.
>>>>
>>>>
>>>>
>>>> "Jonatan Nilsson (BlackMan890)" wrote:
>>>>
>>>>> That is entirely correct but people sometime make the mistake on
>>>>> having
>>>>> the
>>>>> same password for everything which is not such a good idea.
>>>>>
>>>>>

>>>> "Kevin Spencer" wrote:
>>>>
>>>> > Some apps just don't need much of it.
>>>> >
>>>>
>>>
>>>
>>>
>>>
>
>
>
.

• *Follow-Ups:*

- ◆ [Re: Beginning C# Q](#)
◇ From: Kevin Spencer

• *References:*

- ◆ [Re: Beginning C# Q](#)
◇ From: Cody
- ◆ [Re: Beginning C# Q](#)
◇ From: Kevin Spencer

- Prev by Date: [Re: Overriding CollectionBase.Clear\(\)](#)
- Next by Date: [Re: Overriding CollectionBase.Clear\(\)](#)
- Previous by thread: [Re: Beginning C# Q](#)
- Next by thread: [Re: Beginning C# Q](#)
- Index(es):
 - ◆ [Date](#)
 - ◆ [Thread](#)