

## In This Chapter

Filters

Preview

SQL Queries

Groups

Record Tagging

Relationship Tree

### Note

The **View Filters:** field is no longer sticky in GoldMine Premium. This means when the user closes the **Filters and Groups** dialog form, the selected UserID will no longer be saved. The next time this user enters the **Filters and Groups** dialog form, they will enter it displaying the filters of the logged in UserID.

This is another one of those fun chapters for me to write, and in this writing I'm going to incorporate the **Relationship Tree** which, although it has been around for a while, I have yet to really discuss in any of my previous books. Prior to GoldMine Premium, the **Relationship Tree** was known as the **Organization Tree**, but we'll get into more of that later in this chapter.

How useful is the data if you can not easily access that data, or find information contained within the database? It's not very useful at all. Users must be able to cluster the data for mail merges, fax blasts, e-mail blasts, by products purchased, or by any number of various attributes, and GoldMine has five separate ways to perform these sorts of activities.

In this chapter, I am going to introduce you to **Filters**, **Preview**, **SQL Queries**, **Groups**, **Record Tagging** and the **Relationship Tree**, in that order. I say, in that order, as this order is not the same one as is presented on Filters and Groups dialog form shown below in Figure 7-1. I am intentionally presenting these to you in this order as **Groups** could, as well, be built based upon a **SQL Query**, and I thought that it would be best if I covered the **SQL Query** ahead of **Groups** for that very reason.

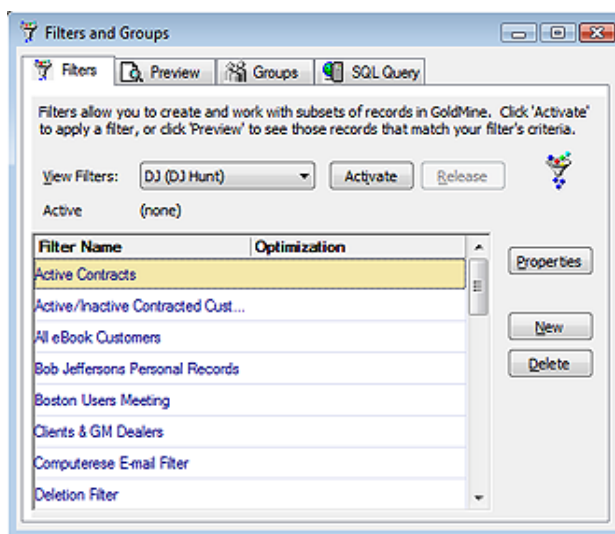


Figure 7-1

What are the distinguishing characteristics of the three methods? Well, **Filters** and **Groups** can be activated, and used, once created, for mail merges, fax blasts, etcetera. On the other hand, the **SQL Query** only pulls data from the different tables into a single query table. The resulting table is normally referred to as a cursor. You can not activate the results of the **SQL Query** and, in turn, use it as you would a **Filter** or a **Group**. You can, as I stated earlier, build a group of records based on the AccountNos in the cursor, but, and more importantly, you could select **Output to ► Microsoft Word**, **Microsoft Excel** or to the **Clipboard** directly from within the cursor. I would point out that there is a similar **Output to ►** option for **Groups**, but it doesn't put out anywhere near the information that can be gathered through a single **SQL Query**. In fact, this option, from within **Groups**, only sends the **Member**, **Sort**, and **Reference** information to Word or Excel. Therefore, in short, **Filters** and **Groups** point to the contact records that contain the information that you are seeking, whereas the **SQL Query** result cursor actually contains the information itself, and not just pointers to records.

What is the difference between a GoldMine **Filter** and a GoldMine **Group**? An excellent question, indeed. They both display records based on some predefined criterion. The filtered information is, however, **Dynamic** while the grouped information is **Static**. Any time a **Filter** is activated against a da-

## Filters

tabase, all of the records that meet the filter condition(s) will be available for various activities that employ filtered sets. On the other hand, **Groups** are a fixed list of records that are created at the time that the individual Group is built. The pointers to the records included in the Group are stored in the **ContGrps** table. This is more of a snapshot in time of which records met your conditions at the time that the Group was created. Yes, one can add to a Group after its has been built, changing that particular Groups point in time. One must remember to perform that step prior to deploying the Group in activities that work with subsets of records. **Filters - Dynamic** versus **Groups - Static**, quite a significant difference in result sets wouldn't you say?

There is another significant difference between **Filters** and **Groups**. Using the GUI, filters can only be built against information contained in the **Contact1** and **Contact2** tables. A group, however, may be generated based on filtered records, previewed records, SQL queried records, scheduled calendar activities, complete history activities, supplemental contact data, tagged records or search results. Wow! Static, though they are, one could build a list of accounts based on virtually any table in the GoldMine database that contains the **AccountNo** field.

As I stated, I will begin with the Filters tab. I like to envision a filter as a sieve. One selects the size of the screen on the sieve ( the condition ), and when the sieve is filled with grains of sand, only those that meet the sieve screen size pass through. All other grains of sand are still in the sieve, however, one can only work with the material on the ground that has made it past the sieve screen.

Let's discuss the basic use of the filter GUI first, and then we can continue on into some more advanced material. Referring to Figure 7-1 on the previous page, the first field that one encounters is the **View Filters**: field. Here, if the user has the proper rights, is a drop-down list of all of the users that GoldMine has within its system, as well as the one group (**public**). Filters are created for a specific UserID or for the (public) user category. Any user, having proper access rights, may employ any filter constructed by any other user. In as much, it is important, when naming a filter, that the creator be cognizant of the fact that other users must be able to distinguish from the **Filter Name** exactly what the filter will accomplish.

**Note**  
I've been meaning to ask FrontRange why they maintain the use of the **dBase Expr.** radio button in the **New Filter** dialog form in GoldMine Premium.

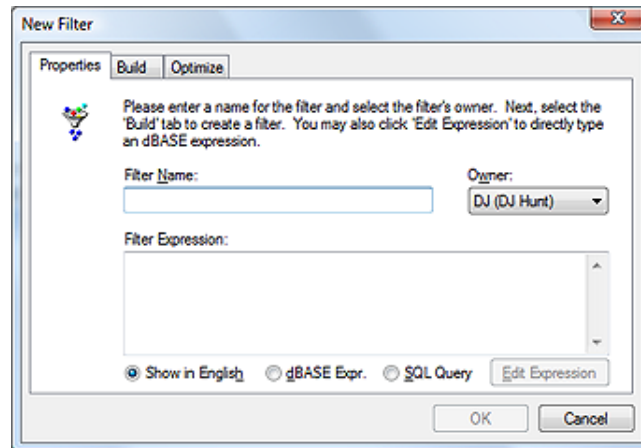


Figure 7-2

Now, click on the **New** button to bring up the **New Filter** dialog form as shown here in Figure 7-2. The first field that is available, is the **Filter Name**: field. This is the field where the user is asked to supply as descriptive a representation of the filter as the allotted space permits. For this example, I would insert into this field, **Created On > 8/1/2009 - CreateOn Field**. I will leave the **Owner**: field set at **DJ (DJ Hunt)**, and, at this point in time, the **Filter Expression**: field will remain empty until we have finished building this new filter.

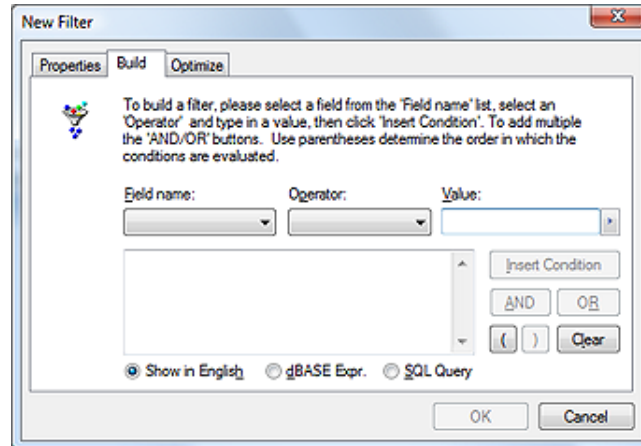


Figure 7-3

Click on the **Build** tab to bring up the dialog form shown here in Figure 7-3. This is the GUI that is utilized to build the filtering condition(s). There are three input fields, the first two are drop-down list, while the third is a text based field. The first list field contains the **Field name**: list. Clicking on the drop arrow to view the list, the user should notice that only those fields that are contained in their **Contact1** and **Contact2** tables appear in this list. For example, the user will not see e-mail address in this list, as the e-mail address is contained in the **ContSupp** table. For this example, I am selecting the **CreateOn** field from the drop list.

In the next field, the **Operator:** field, I will select **Greater or Equal** from its drop list. The user will notice that the operators are in plain English. Here is a list of the available operators in this drop list:

- Equal to
- Not Equal to
- Greater than
- Less than
- Greater or Equal
- Lesser or Equal
- Begins with
- Contains
- Does Not Contain
- Is Empty
- Is Not Empty

**Tip**

*In past versions of GoldMine, particularly the GoldMine Standard Edition, I would have employed the **Optimize** tab, and its ability to speed up filters by using existing GoldMine Indexes. To date, I have found that using this feature in GoldMine Premium has no effect on positive performance, and, conversely, sometimes causes the filter to not function properly.*

*The reader is advised to test this feature against the version of GoldMine that they are using to determine whether its usage is beneficial to their needs.*

*The reader is further advised, if they wish to Optimize their expressions, that they do so by modifying the dBase and SQL expressions utilizing the indexed mirror fields ( refer to Chapter 8 - The Tables ) when possible. All of the indexed mirror fields begin with **U\_** such as:*

Company mirrored to U\_Company

**Note**

*I've been meaning to ask FrontRange why they maintain the use of the **dBASE Expr.** radio button in the **New Filter** dialog form in GoldMine Premium.*

In the **Value:** field, I have entered **8/1/2009**, and at this point, I would click on the **Insert Condition** button which will append to the memo field, in plain English, **Created on is Greater or Equal "8/1/2009"**. I could add more to this expression using the **.and.** or the **.or.** option, but for this exercise let's just click on the **OK** button to save the filter.

In my 4607 record database, if I right-click on this filter to bring up the local menu, and if I select **Count...** from the local menu, the filter returns a result of **Count: 8 (0.2%)**. This count is returned in milliseconds without having employed the **Optimization** feature for the filter. This filter may not work well against older GoldMine installations, however, as the **Contact1.CreateOn** field is a relatively new field to the GoldMine tables. Old time GoldMine users may have a number of records with no data in the **Contact1.CreateOn** field.

Up until recent releases, GoldMine had always derived the create on date by analyzing the **AccountNo** field. After FrontRange incorporated the **Contact1.CreateOn** field into GoldMine, they needed to come up with something that would allow older upgrading users to access the created on date information. The result was the creation of the **accdate()** function ( Refer to Appendix A ). This function can extrapolate the creation date using the **Contact1.AccountNo** field, however, unlike previous versions of GoldMine, in GoldMine Premium you cannot create a dBase expression unless you can create an equivalent SQL where clause. Hence, based on our knowledge of the **Contact1.AccountNo** field ( refer to Chapter 8 - The Tables ), we know that the first 6 characters of the **Contact1.AccountNo** field are the create on date.

Click on the **New** button again. This time, let's give a **Filter Name:** of **Created On > 8/1/2009 Using AccountNo**. Now, instead of clicking on the **Build** tab, select the **dBASE Expr.** radio button, and then click on the **Edit Expression** button. This should bring you into the **Edit Expression** dialog form where you are free to type whatever legal dBase expression you desire. I'll show you other uses for this later.

Here is the expression that I would like you to type into this field:

```
left(AccountNo, 6) >= [A90801]
```

Now, instead of clicking on the **Build** tab, select the **SQL Query** radio button, and then click on the **Edit Expression** button. This should bring you into the **Edit Expression** dialog form where you are free to type whatever legal SQL where clause that you desire.

Here is the where clause as I have it entered:

```
left(C1.AccountNo, 6) >= 'A90801'
```

You may have noticed, although I did not type the **WHERE** portion of the clause, that it still appears in the dialog form, Figure 7-4. The dBase & SQL expressions equate, hence, the **OK** button is enabled, and I can save this filter. Best of all, it too will function as expected. This, however, is not a very dynamic filter as it must be modified each time that you plan to place it in use. I think that I prefer the old ways where you could put in a dBase expression or a where clause, and it would evaluate properly.

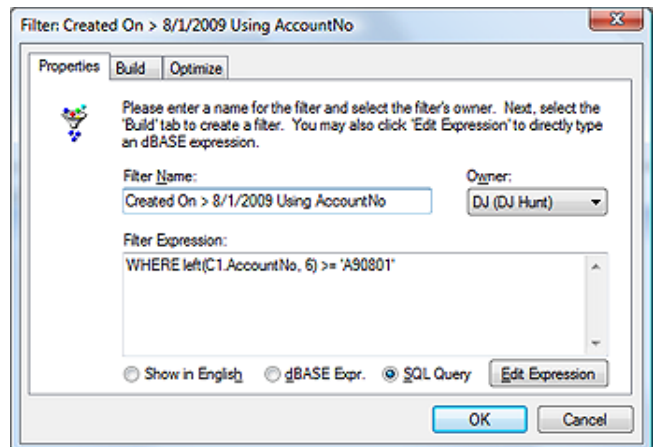


Figure 7-4

Where we could have created more dynamic filters before, we are more hampered in GoldMine Premium unless you are extremely fluent in SQL Query development.

Once completed, click on the **OK** button of the **Edit Expression** dialog form, and then on the **OK** button of the **New Filter** dialog form. Now you should be back to the **Filters and Groups** dialog form, right-click on the new filter and select **Count...** from the local menu. Against my database the results were **Count: 12 ( 0.3% )**, and there was no discernible difference in process speed.

In previous books I was able to address this question from the forum: *"How can I build a filter for all of my contacts that have an e-mail address?"* Alas, GoldMine Premium has taken away this ability to the best of my knowledge. No longer can you utilize:

```
.not. empty(&EmailAddress)

.not. empty(Fax) .and. empty(&EmailAddress)

upper(c1.Phone1) > ' '
and c1.AccountNo not in
(select AccountNo
 from ContHist
 where (OnDate+30) >= getdate())
```

That is, unless you can devise a SQL where clause, or a dBase expression, to match the converse of these items. So let me now show you how this needs to be done in today's GoldMine Premium.

Let's start by creating a **New Filter**, and we'll call it: **Contacts with E-mail Address**. Good. Now let's click on the radio button for  **dBase Expr.**, and then on the **Edit Expression** button. In the **Edit Expression** dialog form, enter:

```
.not. empty(&EmailAddress)
```

You remember that dBase expression, don't you? Go ahead and click on the **OK** button. No change yet from the old days, however, we now need to balance the equation with a comparable SQL Where clause. Let's select the radio button  **SQL Query**, and then on the **Edit Expression** button again. This time, however, in the **Edit Expression** dialog form, we'll enter:

```
C1.AccountNo in
(select AccountNo
 from ContSupp
 where U_CONTACT = 'E-MAIL ADDRESS'
 and Zip like '_1%')
```

Now let's click on the **OK** button on the **Edit Expression** dialog form, and then again on the **OK** button on the **Filter:** dialog form. That's it, we're done. Right-click on the Filter and select **Count...** from the local menu. My count shows: **Count: 3873 ( 84.0% )**, and took less than a second to generate that number.

Here are the expressions for the converse of the above filter, Contacts without a Primary E-mail Address:

```
empty(&EmailAddress)

C1.AccountNo not in
(select AccountNo
 from ContSupp
 where U_CONTACT = 'E-MAIL ADDRESS'
 and Zip like '_1%')
```

My Bad! My editor picked this up. I had stated on the first page that I wanted to include two new sections in this chapter for this book. One was the **Preview** section, while the other was the **Relationship Tree** section. Low and behold, I forgot both sections, hence, the Editors rejection of the manuscript.

Looking back at Figure 7-1, you may have noticed the **Preview** tab in between the **Filters** tab and the **SQL Query** tab. It is here that FrontRange has chosen to enhance the GoldMine Filters. Before we leave the **Filters**, however, let's create two rather simplistic filters to use in this exercise.

The first to select all of my Clients for which I maintain a marker in the **Contact1.Key1** field labeled **Category**.

```
Category (Key1) Contains Client
```

**Note**  
You will not be able to enter Carriage Returns or Line Feeds, and the where clause is shown here formatted for presentation purposes only. Your where clause will be in one continuous line.

## Preview



Now I'm going to create another filter where:

### State Equal to MA

As that filter was the last filter that I created, it remains highlighted ( selected ). While this filter is highlighted, let's choose the **Preview** tab, refer to Figure 7-5. In the upper box you'll notice our filter **In English** as opposed to being **In dbase**, and it is:

### State is Equal to "MA"

Normally, the second box would have been blank at this point, however, for Figure 7-5, I had already clicked up the **Search All** button which delivered 237 records.

Now let's go back to the **Filters** tab, and highlight the **All Clients** filter. Immediately thereafter, switch back to the **Preview** tab, and look at the top box this time. You'll notice that it now contains this filter expression in English.

### Category ( Key1 ) Contains "Client"

However, if I click upon the **Drill Down** button, and I did for Figure 7-6, you'll notice that the expression has changed to:

### (State is Equal to "MA") AND (Category ( Key1 ) Contains "Client")

On top of that you'll want to look at the new **Count: 80** figure. Now I know that this simple filter could have been built from the **Filter** tab, however, take this to the level of a couple of complicated filters, and you have a nice system. Especially when you realize that you can now click upon the **Save** button to update the information in the highlighted filter on the **Filters** tab, or to create a totally new filter utilizing the expression now created.

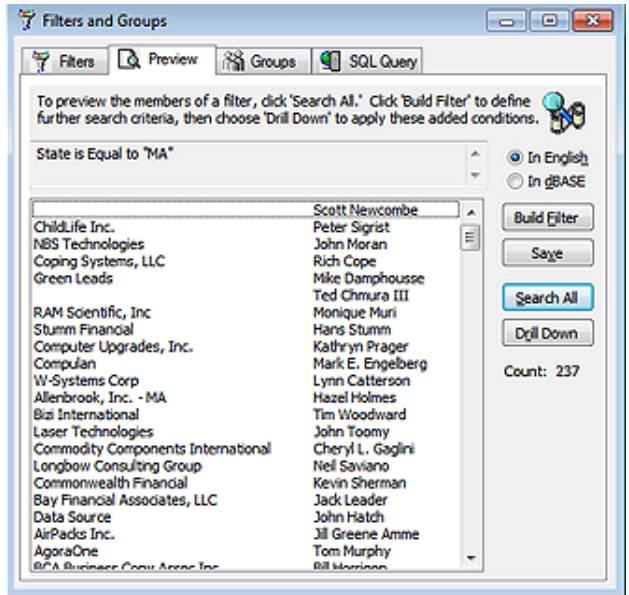


Figure 7-5

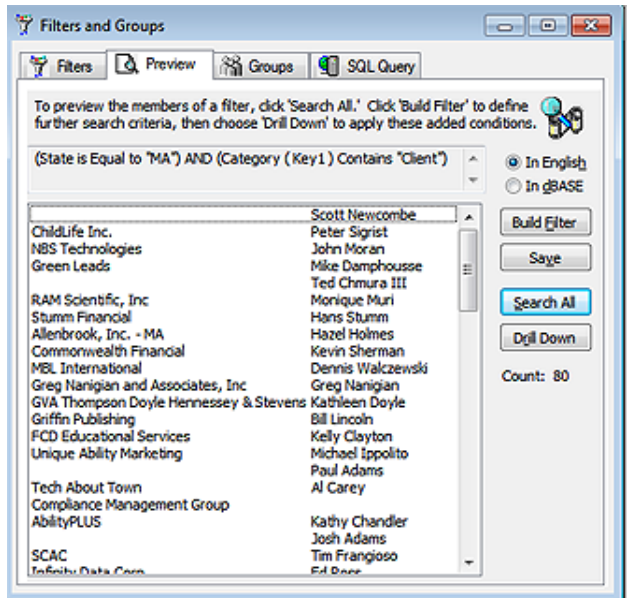


Figure 7-6

Alternatively, you could modify the expression right from the **Preview** tab, by clicking on the **Build Filter** button. WOW! Can it get any easier?

## SQL Queries

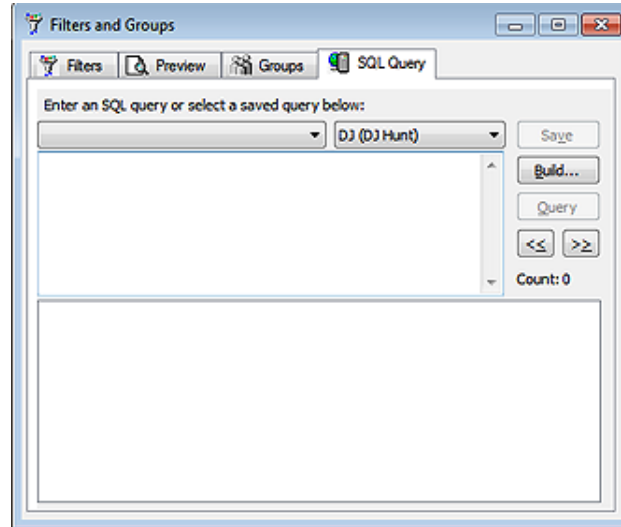
I consider the SQL Query to be the most under utilized gem contained within the GoldMine application. Think about it, people have paid me \$249.00 for my Beyond Gold application ( no longer available ) to export the other **Contacts** from their GoldMine **ContSupp** table when they could have just as easily utilized the **SQL Query** tool in GoldMine. Individuals get fixated on the fact that GoldMine can not export anything except information from the **Contact1** and **Contact2** tables unless exporting to GoldMines own interpretation of XML. They go to the forums and ask, often, *"I have to send a list of other contacts to a mailing house, how can I export this information?"*. Then you see all of the third party solution providers hawking their wares, as I recently saw it stated, and forgetting to even mention that this could just as easily be accomplished within the GoldMine product itself.

By the way, it is my personal policy, when answering questions in the forums, to only give the GoldMine solution answer to the question unless there is no GoldMine solution. Then, and only then, do

I mention the possibility of trying a third party product. Have you noticed that this is a particular sore spot for me?

All right, let's move on then. As a prerequisite to something that I am going to attempt, I would ask you to go to any record, that is in the State of California or Georgia. As an additional requirement, you want to make sure that the record has Additional Contacts. Double-click on any of the Additional Contacts to bring up the **Additional Contact at ...** dialog form. Place an **X** in the  **Merge:** field,

and then click on the **OK** button. We are going to want to use that later.



Good, now that this prerequisite is established, let's start out by saying that you want to send a list of **Additional Contacts**, that are linked to a primary contact record, and where the primary record resides in the state of California or the state of Georgia, to a mailing house. An awkward and possibly unrealistic request I know, but this will serve as the basis for my first example.

Looking here at Figure 7-7, in the medium size box just under the user name, let's type the following:

Figure 7-7

```
select CS.*
from Contact1 C1,
ContSupp CS
where C1.AccountNo = CS.AccountNo
and C1.State in ('CA', 'GA')
and CS.RecType = 'C'
```

Clicking on the **Query** button against my dataset pulled **238** records ( we use to be told the seconds that had evolved, however, in GoldMine Premium this is no longer the case ). Just so that you know, my dataset currently has 4,613 records ( I cleaned up my database since my last book ), so 238 records pulled from that in less then a second is not too shabby. By the way, this and any SQL select statement, can not contain any carriage returns or line feeds. The select statement above contains line feeds, carriage returns, and spaces. As of this writing, GoldMine is very forgiving, and should you accidentally include any of these in your select statement, GoldMine is currently ignoring their inclusion.

So let's talk about this statement a little. To keep my typing to a minimum, I used what is called aliasing of the table names. Look at the **from** clause in the statement above to see how I accomplished this. I stated the actual table name trailed by a space, and then the alias name for the table as, **Contact1 C1**. I am working against two tables, so I separated them in the **from** clause of the statement by a comma ( , ). Now that I have aliased the table names, I can use the alias everywhere else instead of typing out the full table name each time.

My select statement itself, states, display all fields from the **CS ( ContSupp )** table. I used the wild card, the asterisk ( \* ) to accomplish this. I then proceeded to set up some conditions for pulling the information. Conditions, haven't I heard that someplace before? Another name for a condition is a filter. Yes, I am filtering our tables using conditions. The first condition is what is called a simple join clause. I am saying to only look at records where there is a match between the **C1.AccountNo** field and the **CS.AccountNo** field. Remember that the **AccountNo** field is the primary relationship key between most of the database tables within GoldMine.

Now that I have joined the two tables, effectively making them one big table, I can apply additional conditions. In this example, you'll remember, I only wanted the other contacts if the primary contact was from the state of California or Georgia. Therefore, my next condition and **C1.State in ('CA', 'GA')**. Now both conditions, the join and this statement, must return a **True**, based on the Boolean **and**, before the fields will be displayed in the query result window.

Lastly, I needed to add one more condition. The **ContSupp** table contains many records of varying types that are linked to a single contact record. As I am only looking for **Additional Contacts** to the primary record, I am not interested in any **Detail** records, **Linked Documents**, or **Referrals** to name a few of the possible records that could be pulled from the **ContSupp** table. Records in the **ContSupp** table are differentiated by type in the **RecType** field.

**WARNING**  
As a general rule, you should not copy and paste from this book as things like "Variable" have styles associated with them, and we often use line wrapping for better book presentation. Most of this information will not translate well when pasted elsewhere.

All SQL Queries discussed in this book are included in a NotePad created document from which you may copy and paste directly into GoldMine.

**Note**

Refer to **Chapter 8, The Tables** for a complete understanding of the **Cont-Supp** table, and its various **RecType** values.

The following values are possible type values contained in the **RecType** field:

- A Record Alerts
- C Additional contact record
- E Automated Process attached Event
- H Extended detail header
- L Linked document
- O Relationship tree
- P Detail record (formerly known as a Profile record)
- R Referral record

As I am only interested in additional contacts to the primary record, I segregated that set using the and **CS.RecType = 'C'** clause in my statement. For now I ask you to click on the **Save** button, and save this query. This is a very nice feature, unlike the **Groups**, as you will see later, which you have to rebuild from scratch each and every time, GoldMine allows you to save the SQL Query to be utilized again and again as often as you want. I would remind you, when naming your query that you should name it such that, a year from now when you look at that name, you will know from the name exactly what the query does. As always, you are limited in the number of characters, **40** in this case, when naming a query, so use them wisely.

After you have saved the query, let's try to run it against your database by clicking on the **Query** button. Against my database of **37,520** ContSupp records, this query pulled **238** records.

Now I want to change the requirements a little. Everything as before applies, except that I don't want to show all of the fields in the **ContSupp** table. I only want to display the fields necessary to do a mailing. Additionally, I want to make certain that the records in question have an address. What good is a mail list without addresses? This, then, would cause our select clause in the statement to change to the following:

```
select CS.Contact,
       CS.Title,
       CS.Address1,
       CS.Address2,
       CS.Address3,
       CS.City,
       CS.State,
       CS.Zip,
       CS.AccountNo
from Contact1 C1, ContSupp CS
```

This, then, would cause our where clause in the statement to change to the following:

```
where C1.AccountNo = CS.AccountNo
and C1.State in ('CA', 'GA')
and CS.RecType = 'C'
and CS.Address1 is not null
and CS.Address1 > ''
```

Let's save this again, but this time, select to update the current query. Now run this query, and you should have culled your query results down. Against my database of **37,520** ContSupp records, this query pulled **238** records again. I told you that I cleaned up my database before journeying into this book.

Well let's cull it down even further. This is a Christmas Specials mailing that I am sending to my clients through the mailing house. I only want to send this mailing to those that are designated in our database as wanting to receive this type of mailing.

Let's try this now:

```
select CS.Contact,
       CS.Title,
       CS.Address1,
       CS.Address2,
       CS.Address3,
       CS.City,
       CS.State,
       CS.Zip,
       CS.AccountNo
from Contact1 C1, ContSupp CS
where C1.AccountNo = CS.AccountNo
and C1.State in ('CA', 'GA')
and CS.RecType = 'C'
and CS.Address1 is not null
and CS.Address1 > ''
and CS.MergeCodes like '%X%'
```

**WARNING**

Even though **CS.AccountNo** is not required by the mailing house, it is required so as to bind the **Join** properly.

**WARNING**

*If there is a **Note ( Image )** type field in your query result, and if that note contains carriage returns, line feeds or both, then the **Output to ► Excel...** may not be a viable option. **CR** and **LF** in a note, when sent to Excel, could produce some unexpected, and difficult to handle data representations.*

**Note**

*You must be vigilant in your database maintenance. I just ran this query to test it for the book, and found two more records in this state in my system.*

**WARNING**

*In earlier versions of GoldMine Premium, I have noticed that this query does not display the Primary E-mail Address in all cases on the first run. I have found it necessary, for whatever reason, to refresh the Primary E-mail Address switch for the records, and from then on the query has functioned as expected.*

Did I throw you a curve there? I hope not. As there could be many codes in the **20** character **MergeCodes** field, I only wanted the record if one of those merge codes was an **X**. To do that I put in the wild card percent ( % ). By putting one before, and after the **X**, I am stating that I don't care if there is anything to the left of the **X**, and I also don't care if there is anything to the right the **X**. I just want to know if there is an **X** contained anywhere in the **MergeCodes** field.

Running this query, if you followed our prerequisite setup, should result in one record which, by coincidence is how many records I happened to pull. Probably not worth using the mailing house for this few? For this book, however, I will send the name over. The mailing house doesn't have GoldMine, and I can't export this, as the information is contained in the **ContSupp** table. How am I going to get this over to the mail house? The answer is simple. I can get this information over to them in any format that they can handle. I am just going to do it by going through Microsoft Excel first. Obviously, I make the rash assumption that you have Microsoft Excel installed on your computer.

Over your query result table ( the cursor ), right-click anywhere and select **Output to ► Excel...**, and watch what happens. Excel should start, and the query table with the field names should have been populated on an Excel spreadsheet. Everything should be highlighted, so the user would immediately want to go to the Excel menu and select **Format ► AutoFit Column Width** selection in Excel 2007. That will adjust the column widths to show all of the data properly. Now it is simply a matter of selecting the **Save As...**, and then saving the spreadsheet as any file format that your mailing house can handle. The dBase option is no longer available as of Office 2007, however there are many other file types such as .xls, .csv, and more file to which you could save this file. You could then send that file on to the mailing house for their processing as an attachment to an e-mail message. Slick, quick, and you never left your chair.

Now, is there really any need to go out, and to purchase those third party tools to enable exporting from areas in GoldMine that are not part of the **Contact1/Contact2** tables? I think not. I think that you can both save your money, and you can have lots of fun designing your own queries to pull data that your organization can use in various formats. For the rest of this section in this chapter, I will just give you a few queries that are most often requested, and which may be useful to your organization as well. Even if they, themselves, are not useful they may contain elements that will help you in your own design of a **SQL Query**.

I don't know why, but some bug in GoldMine can cause contact records to be assigned more than one Primary E-mail Address. Obviously this would not be good when doing blast e-mails to clients. Here is a select statement that will display the account number, and the number of designated Primary E-mail Addresses for each contact record that has more than one designated Primary E-mail Address account. After running this select, should you have any results, you must take care to assign only one Primary E-mail Address for the specified record.

■ **Contacts - with > 1 Designated Primary E-mail Address**

```
select distinct AccountNo,
count(*)
from ContSupp
where RecType = 'P'
and Contact = 'E-mail Address'
and Zip like '_1%'
group by AccountNo
having count(*) > 1
```

I would like to point out the number of underscores as this is very important, and there is only one. There is one underscore before the **1** as a single character wild card. The % is a generic wild card.

```
and zip like '_1%'
```

The underscore acts as a single character wild card. So in this case, I am stating that I don't care what is in positions 1, 3 & 4 of the **ContSupp.Zip** field, but that the second character must contain a 1. This is another form of wild card usage when building your SQL Queries, and you should keep it in mind as it does come in handy when you are looking for a specific character at a specific position. You could, of course, also have used the **substring()** function to cull out the right character(s).

Now that you have found, and corrected all of those records that had multiple Primary E-mail Addresses, you may want to do an e-mail blast mailing to all of those contact records that have a Primary E-mail Address.

■ **Contacts w/Primary E-mail Address**

```
select distinct
C1.AccountNo,
C1.Company,
C1.Contact,
```



```
CS.ContSupRef+CS.Address1 as 'E-mail'
from Contact1 C1,
ContSupp CS
where C1.AccountNo=CS.AccountNo
and CS.Contact='E-mail Address'
and CS.Zip like '_1%'
```

Here we go with the reverse situation. I need to know all of those records that do not have a Primary E-mail Address. Who knows, maybe you want to mail them a merge letter ( how passé ) instead. Well that select statement is a slight modification of the one above.

■ **Contacts w/o Primary E-mail Address**

```
select AccountNo,
Company,
Contact,
Address1,
Address2,
City,
State,
Zip
from Contact1
where AccountNo not in
(select C1.AccountNo
from Contact1 C1,
ContSupp CS
where C1.AccountNo=CS.AccountNo
and CS.Contact='E-mail Address'
and CS.Zip like '_1%')
```

There you go. All the names in your database that do not have a Primary E-mail Address, and all packaged together nice and neat, ready for your mailing house to process. Well this is really nifty isn't it? I am doing a double select statement again, and as before, I am locating all of those that have a Primary E-mail Address. Notice the not in piece of the statement. If I have a list of those that have a Primary E-mail Address, comparing it to our list of all contacts, then those that are not in the first list but are in the second list must be those without a Primary E-mail Address. Maybe a bit confusing, but after you get a few of these under your belt it'll be natural to you.

Well then, you are in GoldMine Premium Edition so why not combine all of the information from your Primary Contacts and Additional Contacts?

■ **Contacts & Add Contacts w/E-mail**

```
select AccountNo,
Company,
Contact,
Address1,
Address2,
City,
State,
Zip,
Phone1,
(select top 1 ContSupRef+Address1
from ContSupp CS
where CS.AccountNo = C1.AccountNo
and Contact = 'E-mail Address'
and Zip like '_1%'
) as 'E-mail'
from Contact1 as C1
union select CS.AccountNo,
CS.Address3 as Company,
CS.Contact,
CS.Address1,
CS.Address2,
CS.City,
CS.State,
CS.Zip,
CS.Phone,
CS2.ContSupRef+CS2.Address1 AS Email
from Contact1 C1,
ContSupp CS,
ContSupp CS2
where C1.AccountNo=CS.Accountno
and CS.ReclD = CS2.LinkAcct
and CS.RecType='C'
order by C1.Company,
C1.Contact
```

**WARNING**

*I have tested all of these select statements by Copying & Pasting them from this book into my GoldMine Premium Edition of GoldMine, and they have all worked as expected, except where specifically noted.*

*You should do the same, and not attempt to retype them, however, you **must** realize that the single and double quotes are formatted, and must be converted to plain text in the SQL Query analyzer before testing the query.*

Yup, I threw a **union** clause in there to achieve my goal, and, because of the rule that you must have as many fields in the union table as are in the select table, I chose CS.Address3 ( usually blank ) to fill the slot in the union side where there is actually no Company field. So we have now taken all of the Contact information from our database, and placed it into a single cursor from where we can output it to Excel or Word. This is an extremely handy query.

In my next select statement I want to show you another statement that is commonly requested. *“I would like to see all of the records in my database for which there has been no History over a defined period of time.”* In my example, I want those that have had no history within the last 30 days. Here is an example of the select statement for GoldMine Premium that fulfills this request:

### WARNING

GoldMine always likes to see the **AccountNo** field in a query so that GoldMine can synchronize to the Contact record from the resulting cursor. If GoldMine doesn't see the **AccountNo** in your select statement, GoldMine will add it automatically. In a union select this will cause problems unless you add the **AccountNo** to both sides of the union statement.

#### ■ Contacts w/o History in Last 30 Days

```
select Contact,
       Company,
       AccountNo
from Contact1
where AccountNo not in
      (select AccountNo
       from ContHist
       where OnDate + 30 >= getdate())
order by Company,
       Contact
```

Naturally, one might also be interested in those records that have had no history in the last 30 days, and that are not scheduled for anything currently, or in the future. Getting complicated, huh? Let's see if we can compound this.

#### ■ Contacts Not Scheduled, and without History in Last 30 Days

```
select Contact,
       Company,
       AccountNo
from Contact1
where AccountNo not in
      (select AccountNo
       from ContHist
       where OnDate + 30 >= getdate())
and AccountNo not in
      (select AccountNo
       from Cal)
order by Company,
       Contact
```

Where the Contacts w/o History in the Last 30 Days pulled 7,498 records from my database, Contacts Not Scheduled w/o History in Last 30 Days pulled 7,450 records.

All right now, here are two more simple ones. In the older dBase versions of GoldMine, one had an idea of how many records were in their Contact1 table by looking at the **Summary** tab **Record: 2 of 19**. With GoldMine Premium, and with all previous Corporate Edition installations of GoldMine, the same does not hold true. A user can, however, acquire a count against any table with this simple select statement:

#### ■ Count - Records in a Table

```
select count(*)
from Contact1
```

Well, what were you expecting? I did say it was simple. Let's expand on that just a little. How many user defined fields have you created in your GoldMine? Don't know? Don't feel like counting them one by one in the **User Defined Fields** dialog form? Try this select statement:

#### ■ Count - User Defined Fields

```
select count(*)
from ContUDef
where Field_Name like 'U%'
```

In previous releases of the Hacker's Guide series of books, this was where I ended my discussions of SQL Queries, however, as I have learned so much more about SQL Queries since my prior writings, I thought that I would stick a few more queries into this section of this chapter. I needed to develop a query for a client that I thought would be useful for you, my readers. It involves the use of the union clause of the select statement.

**Scenario:** I wanted to select all of the companies for which the company name begins with Computer. In addition, I wanted their Primary Contacts, and all of their Secondary Contacts. Here is the select statement against GoldMine Premium to produce this result set:

■ **Contact/Additional Contacts - from Computer Companies**

```
select Company,
Contact,
AccountNo
from Contact1
where Company like 'Computer%'
Union select Contact1.Company,
ContSupp.Contact,
ContSupp.AccountNo
from Contact1,
ContSupp
where Contact1.AccountNo=ContSupp.Accountno
and ContSupp.RecType='C'
and Contact1.Company like 'Computer%'
order by Company
```

The resulting cursor was composed of some 29 records from my GoldMine database. Figure 7-8 depicts the results of said query against my GoldMine database. As always, at this point, we could select to **Output To ► Excel...**, or to build a Group in GoldMine Premium.

Okay, we all know that GoldMine Premium is the perfect application in that there are no bugs contained there in, however, for some reason many of my clients have come up with duplicate records in their **Contact2** table. We all know that this is a no, no. Each record in the **Contact1** table can have one, and only one, associated record in the **Contact2** table. To find out which contacts have duplicate **Contact2** table records, if any, I would run this query against their GoldMine database:

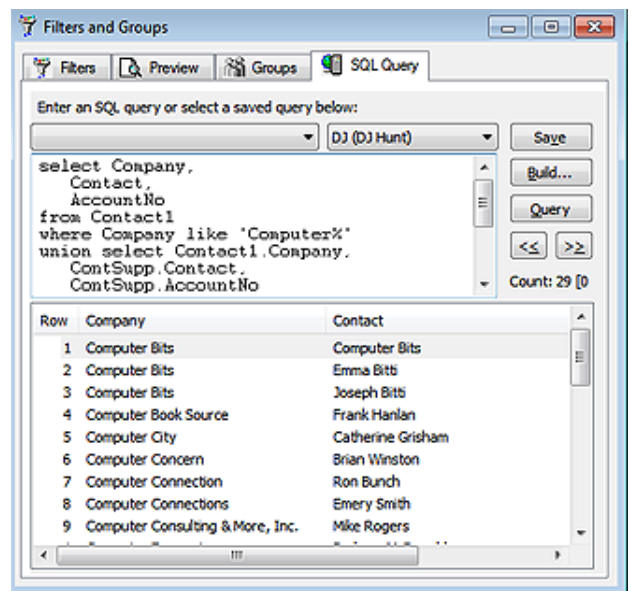


Figure 7-8

■ **Duplicates - Contact2 Records by AccountNo**

```
select count(*),
AccountNo
from Contact2
group by AccountNo
having Count(*) > 1
```

By the way, I just ran this against my **Contact2** table of 4,618 records today, and there were no duplicate records found. I must be getting better at caring for my database. Should your table contain duplicate records, you must immediately rectify the situation. There are a number of different ways of accomplishing the task all of which would require their own book to explain. If you find that you have duplicate records in the **Contact2** table, then you would want to get together with your GoldMine Partner to rectify the matter or you may hire our services if you currently do not have a GoldMine Partner.

Okay, they said that it couldn't be done within GoldMine, but I had always been able to prove them wrong. *"You cannot do an Update Query or a Delete Query from within GoldMine."* Well, John Stillman caught wind of what I was doing, and is no longer permitting **Update** or **Delete** queries from within GoldMine in any form. He deemed that they are too dangerous, and I just have to agree with John on this one. Beginning with GoldMine Premium you can no longer do an **Update Query** or a **Delete Query** from within GoldMine.

Curse you John Stillman for taking away functionality in a program.

**WARNING**

As the GoldMine Administrator, you really do not want any duplicate AccountNo records in your **Contact2** table.

I could not imagine that you would not be interested in some of these other queries, so here are a few more for you to review:

■ **Accounts with Bad E-mail Addresses**

```
select AccountNo,
       ContSupRef+Address1 as [E-Mail Address]
from ContSupp
where (Contact = 'E-mail Address')
and isnull(ContSupRef, '') <> ''
and (select
     case
       when ContSupRef is null
         or charindex('@',ContSupRef + Address1) > 0
         or charindex('.',ContSupRef + Address1) > 0
         or charindex(':',ContSupRef + Address1) > 0
         or charindex('"', ContSupRef + Address1) <> 0
         or charindex('(', ContSupRef + Address1) <> 0
         or charindex(')', ContSupRef + Address1) <> 0
         or charindex(',', ContSupRef + Address1) <> 0
         or charindex('<', ContSupRef + Address1) <> 0
         or charindex('>', ContSupRef + Address1) <> 0
         or charindex(';', ContSupRef + Address1) <> 0
         or charindex(':', ContSupRef + Address1) <> 0
         or charindex('[', ContSupRef + Address1) <> 0
         or charindex(']', ContSupRef + Address1) <> 0
         or right(rtrim(ContSupRef + Address1),1) = '.'
         or charindex(' ',ltrim(rtrim(ContSupRef + Address1))) > 0
         or len(ContSupRef + Address1)-1 <= charindex('.', ContSupRef + Address1)
         or ContSupRef + Address1 like '%@%@%'
         or ContSupRef + Address1 Not Like '%@%.%'
       then 0
       else 1
     end) = 0
```

■ **Calendar - RecType A T O C, for UserID**

```
select *
from CAL
where RecType in ('A', 'T', 'C', 'O')
and UserID = 'DJ'
order by OnDate
```

■ **Casting - Date & Functions**

```
select top 100 cast(LastContOn as varchar),
       datepart(month,LastContOn) [Month 1],
       month(LastContOn) [Month 2],
       LastContOn,
       *
from Contact2
where LastContOn is not null
```

■ **Contact2 - Orphans**

```
select AccountNo
from Contact2
where AccountNo not in
(select Accountno
 from Contact1)
```

■ **Contacts - Not in any Group**

```
select AccountNo,
       Company,
       Contact
from Contact1
where AccountNo not in
(select distinct Accountno
 from ContGrps)
```

■ **Contacts - Record Age = 12 Months Old**

```
select Contact1.Company,
       Contact1.Contact,
       datediff(Month, Contact1.CreateOn, getdate()) as [Age in months],
       Contact1.CreateOn
from Contact1 with (NOLOCK)
left outer join Contact2 with (NOLOCK)
```

**WARNING**

As the GoldMine Administrator, you really do not want any orphaned records in your **Contact2** table.

**Note**

I have introduced a couple of differences here that you should be aware of. Notice the use of the specified join.

Also, notice the use of the square brackets ( **[A]** ) instead of the single quotes ( 'A' ) to delimit a string.

```
on Contact1.AccountNo = Contact2.AccountNo
where (datediff(Month, Contact1.CreateOn, getdate())) = 12)
```

■ **Contacts - w/o Pending Activities**

```
select Contact
from Contact1
where AccountNo not in
(select C1.AccountNo
from Contact1 C1,
    Cal
where C1.AccountNo=Cal.AccountNo)
```

■ **Contacts - All include Primary E-mail Address if Exists**

```
select AccountNo,
    Company,
    Contact,
    (select top 1 ContSupRef+Address1
    from ContSupp as CS
    where CS.AccountNo = C1.AccountNo
    and Contact = 'E-mail Address'
    and Zip like '_1%'
    ) as Email
from Contact1 as C1
```

■ **Count - Accounts by Representative**

```
select upper(Key4) as Representative,
    count(distinct(Company)) as [# Contacts]
from Contact1
where Key1 like '%Client%'
and Company > ''
group by Key4
```

■ **Count - Activites ( Calendar ) by UserID within a Date Range**

```
select UserID,
    Count(AccountNo)
from Cal
where Ondate >= '9/1/2009'
and OnDate <= '9/30/2009'
group by UserID
```

■ **Count - Activites ( History ) by UserID within a Date Range**

```
select UserID,
    Count(AccountNo)
from ContHist
where Ondate >= '9/1/2009'
and OnDate <= '9/30/2009'
group by UserID
```

■ **Count - by State**

```
select State,
    count(State) as Count
from Contact1
group by State
order by State
```

■ **Count - of Cities in State of Massachusetts**

```
select City,
    count(*) as [Count]
from Contact1
where State='MA'
and City > ''
group by City
order by City
```

■ **Count - Mail in Inbox for specific UserID**

```
select UserID,
    count(UserID)
from Mailbox
where UserID = 'DJ'
and Folder = 'X-GM-INBOX'
group by UserID
```



■ **Duplicate - Contacts by Contact/Phone**

```
select count(*),
    Contact,
    Phone1
from Contact1
where Contact > "
and Phone1 > "
group by Contact,
    Phone1
having count(*) > 1
order by Contact
```

■ **Duplicate - E-mail Addresses**

```
select AccountNo,
    Contact,
    ContSupRef,
    Address1
from ContSupp
where ContSupp.Contact = 'E-mail Address'
and ContSupRef in (select ContSupRef
    from Contsupp
    group by ContSupRef
    having count(ContSupRef)>1)
order by ContSupRef
```

■ **Mailbox - Usage**

```
select UserID,
    Folder,
    count(*)
from MailBox
where Folder not in ('X-GM-INBOX',
    'Sent',
    'Filed',
    'X-GM-DRAFTS',
    'X-GM-FOLDERS',
    'X-GM-GROUPS',
    'X-GM-ICALINFO',
    'X-GM-OUTBOX',
    'X-GM-PROP-HTMLTAB',
    'X-GM-HTMLTAB',
    'X-GM-RULES',
    'X-GM-TEMPLATES',
    'X-GM-SUBSENT',
    'X-GM-SUBFILED',
    'X-GM-WEBIMPORT',
    'X-GM-SMIME-CA',
    'X-GM-TD-ITEMS')
group by UserID,
    Folder
order by UserID,
    Folder
```

■ **Notes - GoldMine Premium 8.5.x**

```
select OnDate as [Act Date],
    OnTime as [Act Time],
    Duration,
    Ref as [Reference],
    cast(cast(notes as varbinary(max))as varchar(max)) as Notes
from ContHist
where cast(cast(notes as varbinary(max))as varchar(max)) like '*** DJ%'
```

■ **Records Created within Last 30 days**

```
select *
from Contact1
where (CreateOn > { fn NOW() } - 30)
order by CreateOn
```

■ **Select - 2nd Group of 250 Records**

```
Select Top 250 *
from Contact1
where RecID not in
(select Top 250 RecID
from Contact1
```

**Note**

As of **GoldMine Premium 8.5.x** Notes are now stored as **Images**. Any query statement must account for this, and one must use the `cast()` and `max()` functions to query this information.

```
order by Contact )
order by Contact
```

■ **Select - Convert Function - Specific Type History records on specific date**

```
select Contact1.Company,
       Contact1.Contact
from Contact1 inner join ContHist
  on ContHist.AccountNo = Contact1.AccountNo
where ContHist.sRecType = 'T'
     and (ContHist.OnDate = convert(datetime, '2009-09-11 00:00:00', 102))
     and ContHist.ResultCode like 'C%'
```

■ **Select - Date Function - Contact1 records created within the last 2 days**

```
select *
from Contact1
where (CreateOn > { fn NOW() } - 2)
```

■ **Select - SQL Case Select - If Key1 empty use Key2, if both empty use Key1**

```
select
case Key1
when ''
then Key2
else Key1
end
from Contact1
```

■ **Select - Substring Function**

```
select substring(ContSupp.ContSupRef, charindex('@', ContSupp.ContSupRef)+1, len(ContSupp.
ContSupRef)-charindex('@', ContSupp.ContSupRef))
from Contact1
join ContSupp on Contact1.AccountNo = ContSupp.AccountNo
where ContSupp.Contact = 'E-mail Address'
     and ContSupp.ContSupRef like '%DJ@DJHunt.US%'
```

■ **Select - Substring Function**

Prerequisite: ContSupp.Address1 = [MM/DD/YY]

```
select '20'+substring(Address1, 7, 2)+substring(Address1,1,2)+substring(Address1,4,2) as Date
from ContSupp
where '20'+substring(Address1, 7, 2)+substring(Address1,1,2)+substring(Address1,4,2) >=
'20090101'
     and '20'+substring(Address1, 7, 2)+substring(Address1,4,2)+substring(Address1,1,2) <=
'20091231'
```

■ **Select - Sum/Case Functions - This one is particularly interesting for column display**

```
select UserID,
       sum(case when [FOLDER] = 'X-GM-INBOX' then 1 else 0 end) 'Inbox',
       sum(case when [FOLDER] = 'X-GM-OUTBOX' then 1 else 0 end) 'Outbox',
       sum(case when [FOLDER] = 'X-GM-TRASH' then 1 else 0 end) 'Trash'
from MailBox with (nolock)
group by UserID
order by UserID
```

■ **Top 3 Companies by Sales**

```
select top 3
       C1.Company,
       C1.Contact,
       sum(cast(CH.Duration as Money)) as TotalSales,
       count(CH.Duration) as SalesCount,
       CH.AccountNo
from Contact1 C1
join Conthist CH
  on C1.AccountNo=CH.AccountNo
where CH.RecType = 'S'
     and CH.Duration like '%.%'
group by C1.Contact,
       C1.Company,
       CH.AccountNo
order by sum(cast(CH.Duration as Money)) desc
```

### Note

It is critical that you have the same number & type of fields in your select statements on both sides of the **Union** or you will receive an error when you attempt your query.

#### ■ Union Contact, Sales Pending & History

```
select Contact1.AccountNo,
       Contact1.Company,
       Contact1.Contact,
       Contact1.Key1,
       Contact1.Address1,
       Contact1.Address2,
       Contact1.City,
       Contact1.State,
       Contact1.Zip,
       Cal.OnDate,
       Cal.Ref,
       cast (Cal.Number1 as varchar) as Value,
       cast (Cal.Duration as varchar) as Prob_Result
from Contact1,
     Cal
where Contact1.AccountNo = Cal.AccountNo
and Contact1.Key1 = 'Client'
and Cal.RecType = 'T'
union select Contact1.AccountNo,
           Contact1.Company,
           Contact1.Contact,
           Contact1.Key2,
           Contact1.Address1,
           Contact1.Address2,
           Contact1.City,
           Contact1.State,
           Contact1.Zip,
           ContHist.OnDate,
           ContHist.Ref,
           ContHist.Duration,
           ContHist.ResultCode
from Contact1,
     ContHist
where Contact1.AccountNo=ContHist.AccountNo
and Contact1.Key1 = 'Client'
and ContHist.sRecType = 'T'
order by Contact1.Contact,
        OnDate
```

#### ■ Union Query - Information from Contact1 & ContSupp

```
select Company,
       Contact,
       AccountNo
from Contact1
where Company like 'Computer%'
Union select Contact1.Company,
           ContSupp.Contact,
           ContSupp.AccountNo
from Contact1,
     ContSupp
where Contact1.AccountNo=ContSupp.Accountno
and ContSupp.RecType='C'
and Contact1.Company like 'Computer%' order by Company
```

That should give you enough queries to show you the endless possibilities. This, then, brings us to the conclusion of the **SQL Query** section of this chapter. However, you should understand by now that the **SQL Query** is a very, very powerful reporting tool. I suggest that you bone up on your **SQL Query** language, as it can only help you. I found a book from Sams' publishing, called *Teach Yourself SQL in 24 hours* by Ryan K. Stephend and Ronald R. Plew, to be extremely useful, and enlightening.

It is important to understand that a SQL Query is only the returning of data which exists in your database, and that it, in and of itself, cannot be activated or used as would a Filter or Group. Yet, the results of your SQL Query could be utilized to create a Group.

Let's say, as I recently had done, that you needed to know everyone in your GoldMine database that was still using GoldMine Standard Edition, and that had not, as yet, upgraded to GoldMine Corporate or Premium Edition. As I store this information as a **Detail** ( **ContSupp** table ), against my GoldMine database this would be the resulting query to pull the AccountNos for those records:

```
select distinct Contact1.AccountNo
from Contact1,
     ContSupp
where Contact1.AccountNo=ContSupp.AccountNo
and Contact1.MergeCodes like '%E%'
```

## Groups

and Contact1.MergeCodes not like '%N%'  
and ContSupp.ContSupRef like '6.%'  
and Contact1.AccountNo not in  
(select ContSupp.AccountNo  
from ContSupp  
where (ContSupp.ContSupRef like '7.%'  
or ContSupp.ContSupRef like '8.%'))

**Note**

Key phrasing here: a SQL Query definition can be saved for later use, whereas, **Groups** must be rebuilt each and every time. There is no saving of the **Groups** definitions as there is for the **Filters** or the **SQL Query**.

**Tip**

Keep in mind, as I am building a Group from the resulting cursor, that I only require the AccountNo. When building Groups only the AccountNo is required, and only one AccountNo will be added to the Group regardless of how many times it appears in the query results.

After all, do you really want to eBlast two of the same e-mails to the same Contact? I think not.

**WARNING**

If this task is to be successful, you must go directly from the **SQL Query** tab to the **Groups** tab immediately following the clicking of the **Query** button as shown in Figures 7-9 & 7-10.

That query pulled some 171 records from my GoldMine database today, and, more importantly, that query can be saved using the **Save** button, Figure 7-9, and reutilized again and again.

The reader will remember that a **Group** represents the data in GoldMine at a specific point in time. That point in time is the date, and the time that the group is generated. There are two grids on this dialog form under the **Groups** tab, Figure 7-10. The top grid, with the **Group Name**, **Code**, and the **Members** fields is where the user adds their Groups. As I mentioned earlier in this chapter, Groups are stored in the **ContGrps** table, and they are associated with the UserIDs. A right-click in this grid area will bring up a local menu from which the user may **Select User...**, from the GoldMine list of users, for which they may wish to employ or examine that UserIDs defined Groups. That, however, is only there for legacy as GoldMine Premium 8.5.x now has a **View Groups:** of UserID option from where one could just click on the drop arrow to select any of the defined UserIDs within GoldMine. More simply put, any user may use the groups of any other user if they have the **Access Rights** to do so.

It is also from this local menu that the user may select to create a **New...** group or by simply clicking upon the **New Group** button. Doing so will bring up the **New Group** dialog form

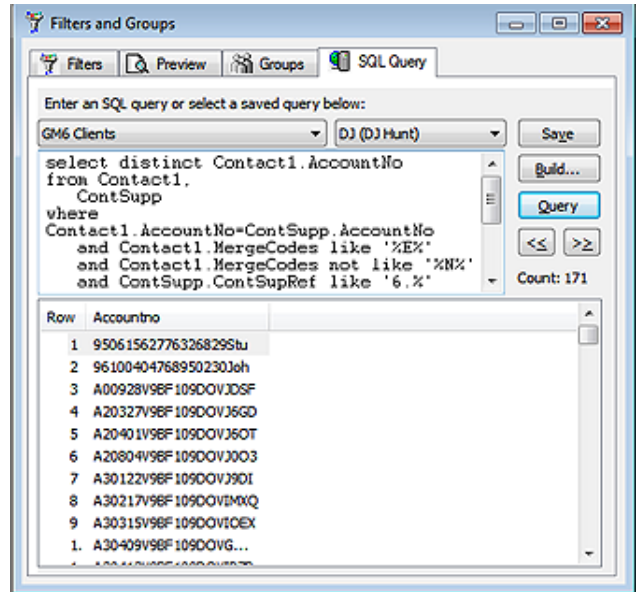


Figure 7-9

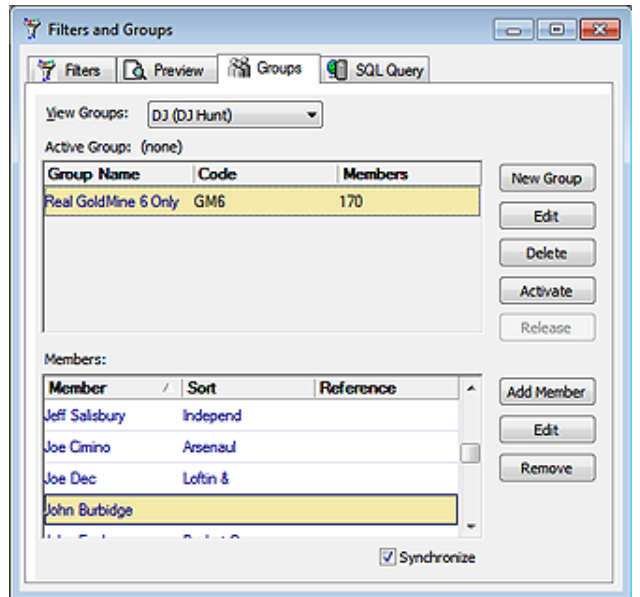


Figure 7-10

shown here in Figure 7-11. The creator is asked to supply a **Group Name:** in that field, however, the creator is not allocated much space to give the Group a really descriptive name. The creator is limited to only 24 characters with which to define this Group.

The **Code:** field, not to be confused with the **Activity Code** field when scheduling activities, is strictly a **Sort Order** field. GoldMine does not alphabetize the Group list by the **Group Name:** field, but instead, sorts the Group list alphabetically based on the **Code:** field. Additionally, as I stated earlier, not only may one view the Groups of another user, but they may also construct a Group in another users name. From the **User:** field

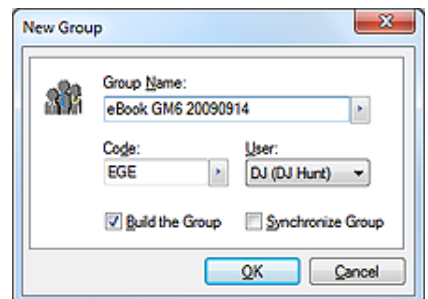


Figure 7-11

**Tip**

When naming anything anywhere in GoldMine, try to give a name that will describe what the action does. You want to make certain that, when you read the description a year from now, you, or anyone else, will be able to understand clearly what the Group consists of. As Groups are a static representation of data at a specific date/time, it would not be a bad idea to identify that date as a string date in your **Group Name:** description as I have done in Figure 7-11.

drop-down list, select the name of the user for which the Group is being constructed. The default item in the drop list is the currently logged in GoldMine UserID, however, many organizations will simply build Groups under the (public) user.

The  **Build the Group** checkbox is select in the default state. If this checkbox is selected, and the user were to click on the **OK** button, GoldMine would bring up the **Group Building Wizard** dialog form immediately, from which the user may construct the desired Group.

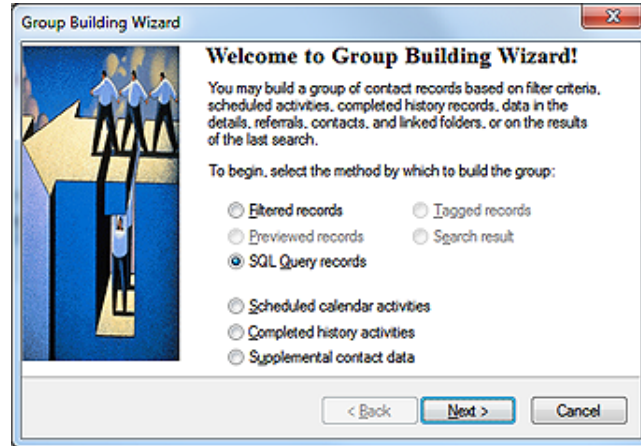


Figure 7-12

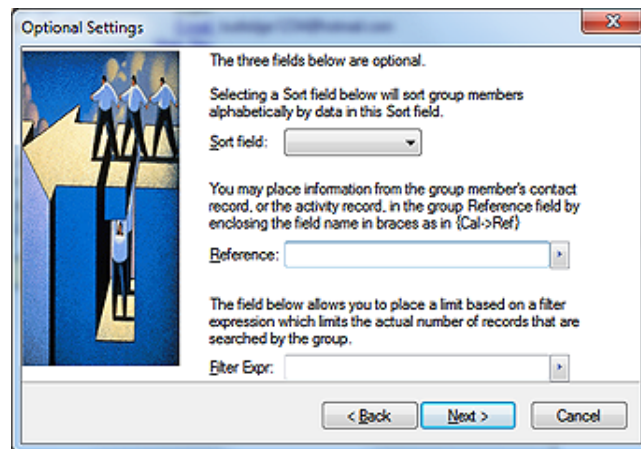


Figure 7-13

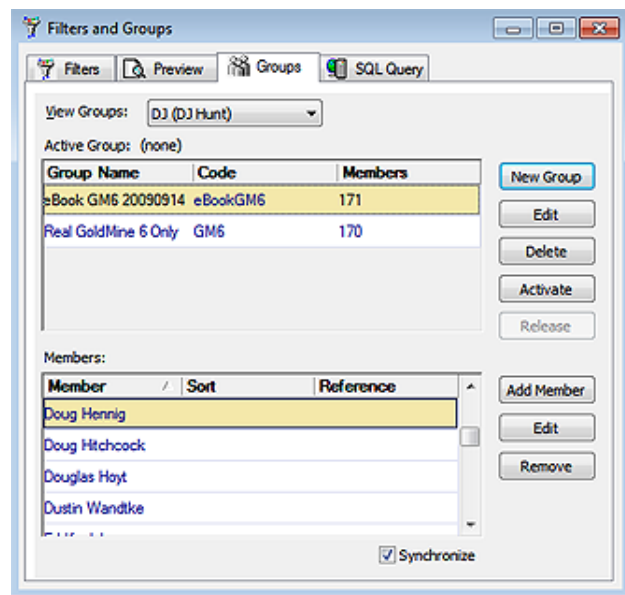


Figure 7-14

Groups are not normally synchronized to remote users via GoldMine in the default state. Should it be your wish, that your remote users have available the Groups which are created Server-side, then the creator of the group must select  **Synchronize Group**. By default, this option is not selected, and one must be cognizant of this fact if they wish to have Groups synchronize to their Remote-side users.

Once the user has completed this dialog form, they must click on the **OK** button. I'll assume that you maintained the default setting, and will go directly into the **Group Building Wizard** as shown in Figure 7-12. Yes, I know, I have already preselected the **SQL Query records** for this exercise, however, we all know that in this dialog forms default state, none of these options would have been selected. We are, after all, attempting to build a Group based on our previous SQL Query cursor, and these are the steps to accomplish that task.

Clicking on the **Next >** button will bring up the dialog form shown in Figure 7-13. I will discuss this dialog form in more detail coming up when I discuss something other than creating a Group as a result of your SQL Query cursor. Clicking on the **Next >** button will bring up the **Finish** dialog form, not shown.

Now, clicking on the **Finish** button would start the process, and GoldMine would build the Group from the SQL Query cursor. The **Filters and Groups** dialog form, at least against my GoldMine database, would look like this one shown here in Figure 7-14. It's just that simple, so even though you cannot save your Group building profile, you can save your SQL Queries, and rebuild your Group at any time in the future to capture the latest static information. Once built,



you could activate said Group, and utilize it for Reporting, eBlasting, Merge Documents, etc.

I would like to step back a bit now, and show you, what I probably should have shown you first, how to build your Groups by hand.

For this exercise, we will be building a Group that will match another of my, previously defined, select statements. I want all of the **Contact** records that have a **Primary E-mail Address**. To do this I must build the group based upon **Supplemental contact data**, refer back to Figure 7-10 on the previous page. Before I do this, I would like to emphasize the various options by which one can build a group of contact records.

Groups could be built based on:

- Filtered records
- Previewed records
- SQL Query records
- Tagged records
- Search result
  
- Scheduled calendar activities
- Completed history activities
- Supplemental contact data

Once the user clicks on the **Next >** button, they will be stepped into the dialog form shown here in Figure 7-15.

**Details** will be selected by default, however, as mentioned in **The Tables** chapter, there are other record types maintained in the **ContSupp** table. The user, at this point, could as easily have chosen  **Document Links**,  **Additional Contacts**, or  **Referrals**, all of which represent supplemental contact data. I, however, want the default setting for this exercise.

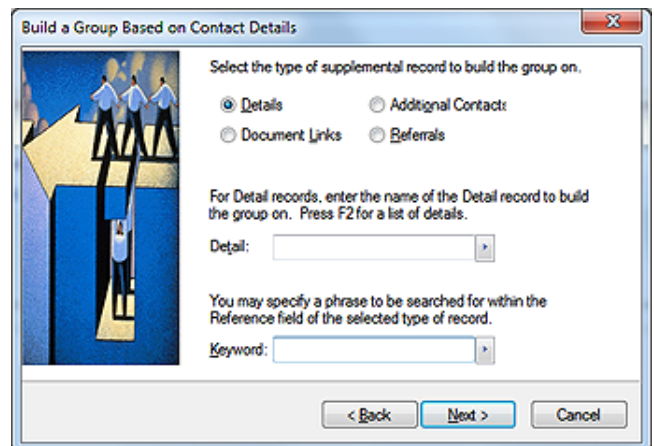


Figure 7-15

The **Detail:** field F2 Lookup list is comprised of all of your defined **Details**. One could type into this field anything that they desire, however, unless it was an item from the F2 Lookup list, there would be little likelihood of finding any records that match the criterion specified. Hence, GoldMine will Force a match against the F2 Lookup List. For this exercise I chose E-mail Address from the F2 Lookup list.

For this exercise, there is nothing that I wish to enter into the **Keyword:** field. I, therefore, leave it blank, and simply click on the **Next >** button to bring us to the dialog form shown here in Figure 7-16. However, you should be aware that the **Keyword:** field search is a contains ( \$ ) statement so that you could have culled your results a little by including something like:

@DJHunt.US

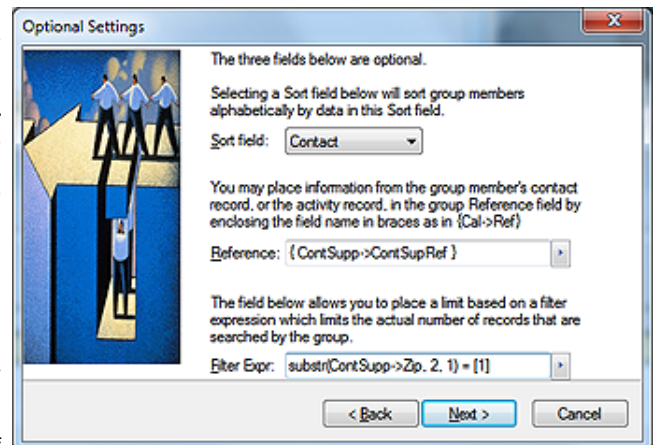


Figure 7-16

The reader will notice that I have selected from the drop-down list of the **Sort field:** field, the **Contact** field on which to sort. The creator of the **Group** may select from any field contained within the **Contact1/Contact2** tables ( refer to sidebar Note ).

The next field that one encounters is the **Reference:** field. If you will refer back to Figure 7-10, and look at the second grid, to the bottom of the figure, you will see columns for **Member**, **Sort**, and

**Note**

No matter which field the creator of the Group selects to sort upon, only the first 8 characters of the selected field will be employed when creating the sorted group list.

## Record Tagging

**Reference.** Whatever expression the user enters here in the **Reference:** field, will be displayed in the **Reference** column of that grid. By default, GoldMine enters **{ContSupp->ContSupRef}**, which is the field that contains most of the actual e-mail address, therefore, I will accept the default in this case.

The last field on this page of the wizard, is the **Filter Expr:** field, and it is here that I must qualify those records that I wish in my list. Back in my **SQL Query** select statement, for this same Group, you'll remember that I discussed that the **Primary E-mail Address** is identified by having a **1** as the second byte of the **Zip** field in the **ContSupp** table ( refer to **The Tables** chapter ). As I am interested in only those **Contact** records that have a **Primary E-mail Address**, I must have a filter expression that will establish this same criterion. I have entered:

```
substr(ContSupp->Zip, 2, 1) = [1]
```

Once entered, I would click on the **Next >** button, and then on the **Finish** button. This produces the group result of 336 records when built against my GoldMine database.

Has anyone noticed a drawback in our discussion of the **Groups** tab, or the creation of a group? Not anywhere, in the wizard, did it ever ask me to save the profile. I've mentioned this earlier in this chapter. Each time I need to **Add Members**, or need to **Build a Group**, it must be done from scratch, stepping through the wizard each time. I must remember the fields that were previously selected, and the expressions that were used each time that I want to rebuild the **Group** to refresh it. In this exercise, it would have been wiser to have performed my **SQL Query** first, which could be a saved profile, and then to have built the **Group** based upon the resulting query cursor. It is much easier, and significantly faster. Hence, the reason that I covered **SQL Queries** before **Groups** in this chapter.

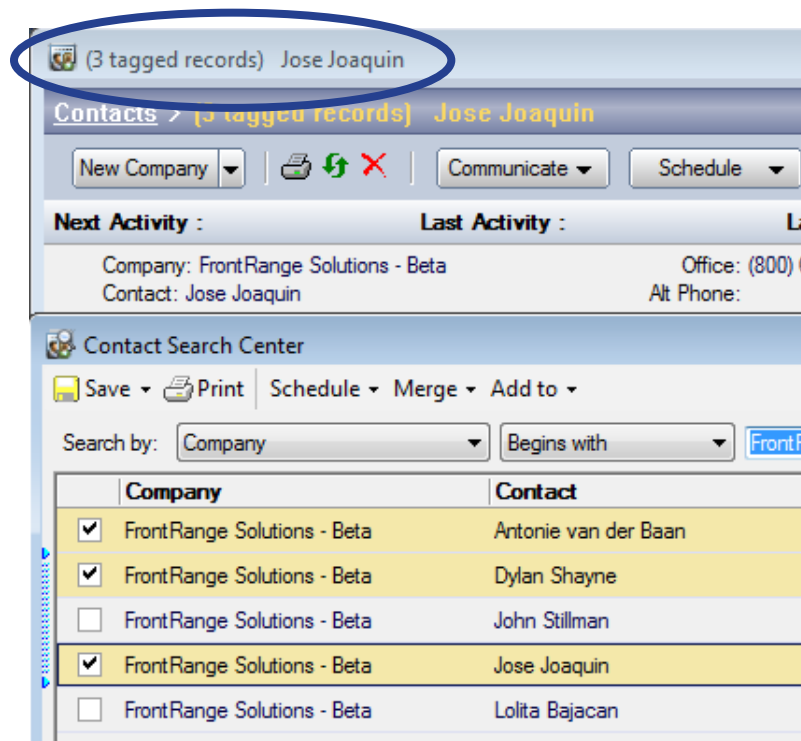


Figure 7-17

Groups may also be built based on **Tagged records**, therefore, I thought that I would take a moment to explain how one goes about **Tagging** a record. As shown here, Figure 7-17, you should bring up the **Contact Search Center** dialog form. Holding the **Ctrl** key down or simply check the box preceding the record to select ( tag ) a record. Notice that, in the **Contact** dialog form behind, the title has changed to include, **1 tagged records** when you select the first **Contact** record. You may tag as many records as you wish by finding the name in the **Contact Search Center**, holding the **Ctrl** key down, and selecting ( left-clicking ) the record with your mouse or by simply checking the box preceding the record. You will notice that my titlebar shows that I have **3 tagged records**, and, as they are all on the same **Contact Search Center** screen, you can see the check marks next to those that I have tagged. You must leave the **Contact Listing** window open until after you have built your **Group** based upon **Tagged records**.

## Relationship Tree

You may also use these tagged records just as they are without building a **Group**. That's right, once **Tagged**, you may use this as your **Active Filter** for reports, merge documents, and e-mail merges or just anything that utilizes a **Filter/Group**. This will be your **Active Filter** until you close your **Contact Search Center**.

We've talked about the **Relationship Tree** elsewhere in this book on a number of occasions, but did you realize that a well structured Relationship Tree can also be used as an **Active Filter**? That's right. Take a look at this **Front-Range** tree in Figure 7-18. Notice that I am only showing you two of my many FrontRange Sub Folders, however, this should be enough to illustrate my point.

For argument sake, let's just say that I wanted to eBlast to everyone at FrontRange. I would simply highlight the **Book** level folder ( **FrontRange** ), And then right-click on that highlighted branch to bring up the local menu. From that local menu, I would then select **Activate Relationship Tree** or you could simply click on the **Activate** button on the toolbar running across the top of the relationship tree list.

Notice the titlebar in Figure 7-18 which currently reflects the Active Contact record belonging to **Kevin Smith**. After one has activated a tree, or branch of a tree, the titlebar would now indicate, per my example: **Section: Front-Range; Steve Salas**. Now go ahead and release that **Active Filter** by again selecting the local menu, but this time selecting **Release** from the local menu or the toolbar across the top of the relationship tree list.

Let's look at another data gathering feature of the Relationship Tree while we are here. Prior to the Relationship Tree ( Organization Tree ), people would enter all of the contacts belonging with a single organization under the **Additional Contacts ( Others )** tab. This way all of the history for one organization, regardless of who it was with, would appear under the one **History** tab on that record. With the introduction of the Relationship Tree, then called Organization Tree, FrontRange asked us to begin the move to an Accountcentric solution by creating a single contact record for each organization member, and then relating them via the Relationship Tree.

Good enough, but then people wanted to know: What about the corporate history? It would now all be individual history. No problem on the SQL backend. Simply highlight, let's say, the **Corporate** branch of my tree. This will take you to the linked record for that branch. Now click on the **Rollup** button in the toolbar or **Roll up ALL Section's Contacts** from the local menu, your choice. Gold-Mine will now rollup all of the history for all contact records in the branch under the **History** tab of the **Active Contact** record. Sweet!

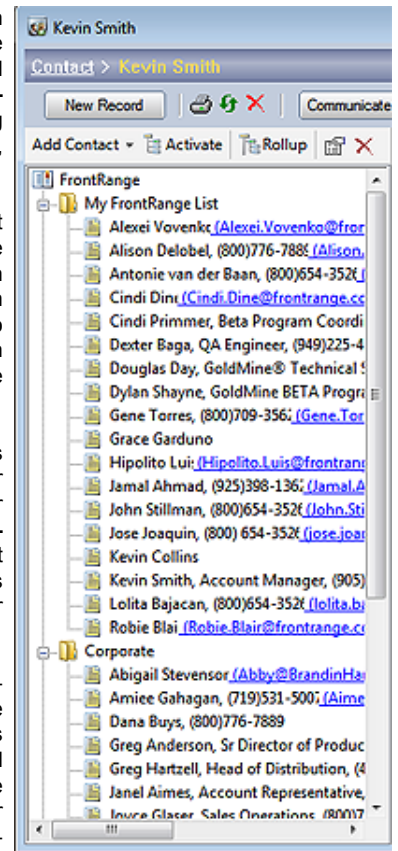


Figure 7-18

