relātia
software corporation

its not in the manual

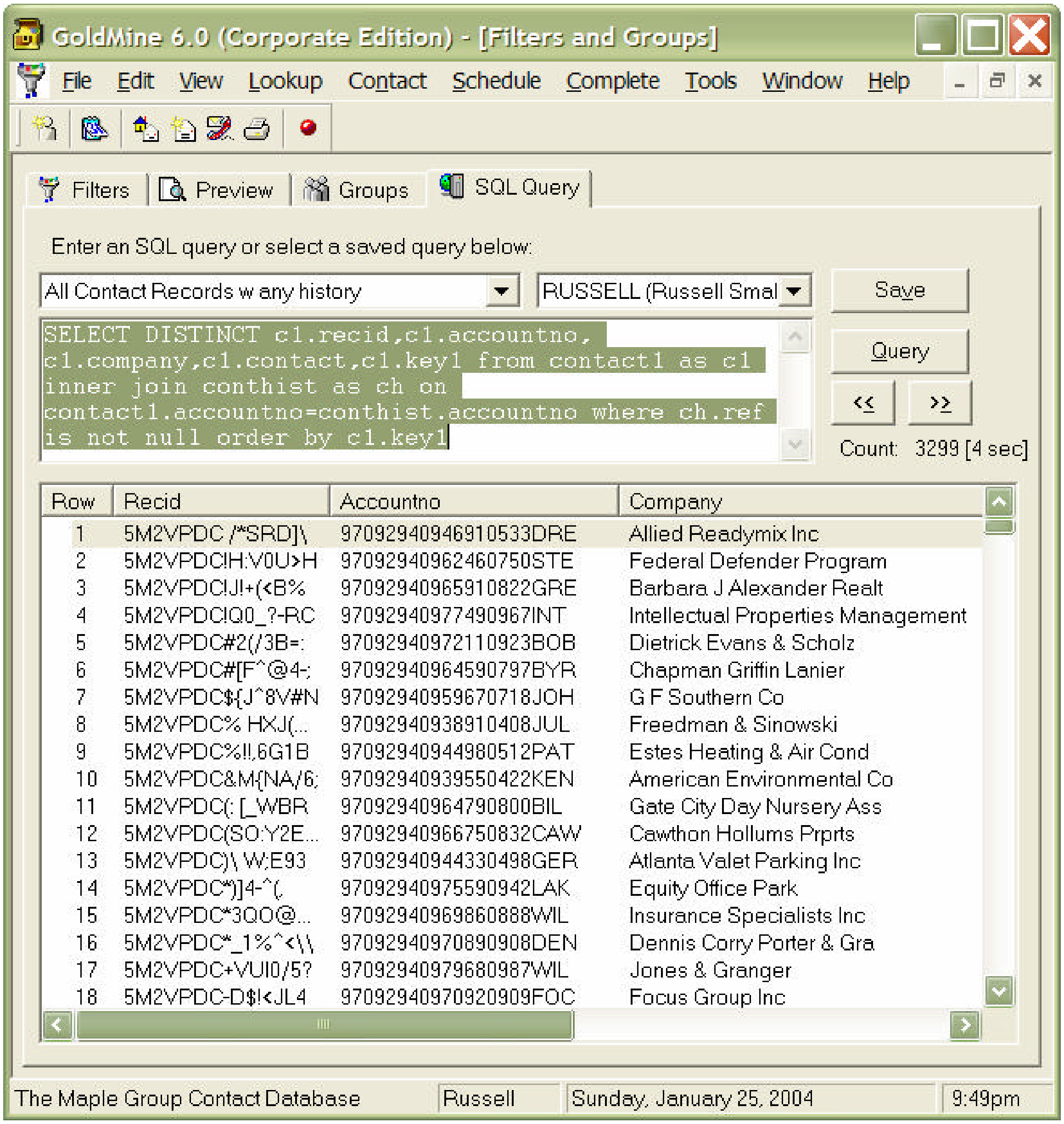
An SQL Cookbook.

The Back-story:

I've always liked the technical cookbook concept. For those of you who aren't familiar with the format, it's basically a loosely structured compendium of ideas based on solving common programming problems. If you have the time, reading through one of these can produce creative lighting strikes or, if you're lucky, help you solve a problem you currently happen to be grappling with.

I thought it might be interesting to write an SQL Query cookbook for those of us fortunate enough to struggle with GoldMine's unique database structure on a daily basis. If you're new to SQL and GoldMine, you might take a look at GoldMine's technical article entitled "Working with SQL Statements" which can be found in the support section of the FrontRange website. It covers the basics and provides a succinct introduction to using the SQL language and specifically how to apply it to GoldMine's table structure.

What follows is a random smattering of semi-interesting SQL Queries that I hope may give you some ideas the next time you're trying to tackle a difficult problem.


GoldMine 6.0 (Corporate Edition) - [Filters and Groups]
File Edit View Lookup Contact Schedule Complete Tools Window Help
Filters
Preview
Groups
SQL Query
Save
Query
<<
>>

Enter an SQL query or select a saved query below:

All Contact Records w any history

RUSSELL (Russell Smal

```
SELECT DISTINCT c1.recid,c1.accountno,
c1.company,c1.contact,c1.key1 from contact1 as c1
inner join conthist as ch on
contact1.accountno=conthist.accountno where ch.ref
is not null order by c1.key1
```

Count: 3299 [4 sec]

Row	Recid	Accountno	Company
1	5M2VPDC /*SRD]\	97092940946910533DRE	Allied Readymix Inc
2	5M2VPDC!H:V0U>H	97092940962460750STE	Federal Defender Program
3	5M2VPDC!J!+(<B%	97092940965910822GRE	Barbara J Alexander Realt
4	5M2VPDC!Q0_?-RC	97092940977490967INT	Intellectual Properties Management
5	5M2VPDC#2(/3B=:	97092940972110923BOB	Dietrick Evans & Scholz
6	5M2VPDC#[F^@4-;	97092940964590797BYR	Chapman Griffin Lanier
7	5M2VPDC${J^8V#N	97092940959670718JOH	G F Southern Co
8	5M2VPDC% HXJ(...	97092940938910408JUL	Freedman & Sinowski
9	5M2VPDC%!!,6G1B	97092940944980512PAT	Estes Heating & Air Cond
10	5M2VPDC&M{NA/6;	97092940939550422KEN	American Environmental Co
11	5M2VPDC(: [_WBR	97092940964790800BIL	Gate City Day Nursery Ass
12	5M2VPDC(SO:Y2E...	97092940966750832CAW	Cawthon Hollums Prprts
13	5M2VPDC)\ W;E93	97092940944330498GER	Atlanta Valet Parking Inc
14	5M2VPDC*)]4-^(,	97092940975590942LAK	Equity Office Park
15	5M2VPDC*3QO@...	97092940969860888WIL	Insurance Specialists Inc
16	5M2VPDC*_1%^<\\	97092940970890908DEN	Dennis Corry Porter & Gra
17	5M2VPDC+VUI0/5?	97092940979680987WIL	Jones & Granger
18	5M2VPDC-D$!<JL4	97092940970920909FOC	Focus Group Inc

The Maple Group Contact Database | Russell | Sunday, January 25, 2004 | 9:49pm

GoldMine's SQL Query Window

First, Some General Observations:

Before we get too far along, I might mention a couple of idiosyncrasies that you should be aware of if you're going to spend any time working with GoldMine's SQL Query feature.

First; As a courtesy to SQL DBA's everywhere, GoldMine limits the result sets returned to 10,000 records by default. If you're going to use this feature on a regular basis, you'll probably want to change that, which you can do by entering the following line to your *username.ini* file:

```
[GoldMine]
SQLQueryLimit=nnnn
```

Where "nnnn" represents the maximum number of records to be returned for any query. I usually set mine to 1000000 (one million). By the way, you should already see the [GoldMine] in the ini file; just add the SQLQueryLimit statement to it.

Next, you need to be careful when porting Queries from one platform to another. You can actually write queries that function perfectly when you're connected to the SQL Server via a "networked" copy of GoldMine but fail when run on a dbase installation. This is because of the subtle differences between the database driver implementations native to the BDE.

You should also be careful when joining tables from the GMBase Directory with tables in the database directory. In Dbase, you'll need to explicitly specify the file paths like this:

```
SELECT
        c1.accountno,
        c1.company,
        cal.accountno,
        cal.ondate,
        cal.ref
FROM
        'C:\Program Files\GoldMine\MyDb\Contact1.dbf' as c1
INNER JOIN
        'C:\Program Files\GoldMine\GMBase\Cal.dbf' as cal
ON
        c1.Accountno=Cal.Accountno
```

Which brings me to my final point(s). You'll notice that I'm explicitly calling for the accountnos in the column selections for each table involved in my join. This is very important habit to get into due to the fact that, if you don't, GoldMine will insert them without your permission and use the wrong alias's (it uses the table names by default) and you'll get an SQL error. Also, I'm going to use the format above for my SQL Query examples In this document for clarity, however, GoldMine's SQL Query Window doesn't provide much text formatting functionality so most of the time, your queries will actually look like this:

```
SELECT c1.accountno, c1.company, cal.accountno, cal.ondate, cal.ref FROM
'C:\Program Files\GoldMine\MyDb\Contact1.dbf' as c1 INNER JOIN  'C:\Program
Files\GoldMine\GMBase\Cal.dbf' as cal ON c1.Accountno=Cal.Accountno
```

after you get them typed in (although the query window will retain formatting if it comes from a cut and paste).

Note: *As with most SQL interpreters, Borland's version has its own syntax rules. You'll need to scour your hard driver for the file **localsql.hlp** which sometimes gets installed in your BDE directory (sometimes not so you'll have to keep looking or contact me for a copy) which is the help file associated with Borland's SQL interpreter and contains details on all the supported SQL functions and the supported syntax.*

Okay, that's all for housekeeping; let's get started.

IN(NOT IN):

We'll start with something easy.

One of my customers asked me the other day if there was a way to get a list of all the accounts with any history. They wanted to prune their database and get rid of contact records who were seemingly just taking up space and had no history items. I don't know of any way to do this with any of GoldMine's built in tools. I suppose you could do a contact history report, but that would be a static document, not to mention limited to just those customers *with* history. We want the ones without. This, of course, is trivial with the SQL Query feature.

This is also a great example of using the IN(NOT IN) clause which, in my opinion is one of the most useful parts of the SQL language when it comes to whipping off one-liners.

Let's take our customers request. Starting with this:

```
SELECT
        Accountno,
        Company,
        Contact,
        Phone1,
        City,
        State,
        Zip
FROM
        Contact1
```

Which will return all the contacts in the contact1 table, we can add the simple line:

```
WHERE
        Accountno
NOT IN
        (Select accountno from conthist)
```

And there you have it. All the contact records in the database with zero history. Pretty simple huh? Of course, you can now build a group from the results or, export the list to Excel or whatever.

By the way, you'll notice (if you actually run this) that it takes forever. Well, okay, the BDE is not the most efficient SQL Query parser in the world, particularly when used in conjunction with Dbase tables, however, have patience, you will get your results back. I'm also making the assumption that you know better than to run queries against your production SQL Server that may bring it to its knees during peak work hours……..

Let's say you want to go the other way, you want all the contacts with history (maybe you want to send a mailer or something, I don't know). In any case, you can just switch the "NOT IN" to "IN":

```
SELECT
        Accountno,
        Company,
        Contact,
        Phone1,
        City,
        State,
        Zip
FROM
        Contact1

WHERE
        Accountno
IN
        (Select accountno from conthist)
```

And there you go.

You can modify this in a number of useful ways…i.e. –

```
SELECT
        Accountno,
        Company,
        Contact,
        Phone1,
        City,
        State,
        Zip
FROM
        Contact1
WHERE
        Accountno
IN
        (select accountno from conthist where ref like 'Billing Detail%')
```

Which returns all contact records who have a history record where the ref says "Billing Detail" at the beginning. (In my company's case, this would be anyone we've done any work for in the past as we use our Time and Billing Software Relatia Time and Billing to capture filed work done for our customers. You can read more about that here: www.relatia.net)

You get the picture, by changing the sub query (select accountno from conthist where blah blah blah) you can slice and dice your contact database in regards to their historical properties any way you like. The same applies to detail records or calendar items. Consider the following:

```
SELECT
        Accountno,
        Company,
        Contact,
        Phone1,
        City,
        State,
        Zip
FROM
        Contact1

WHERE
        Accountno
IN
        (Select accountno from contsupp where contact like 'Web Site%')
```

This would return (I hope you're getting the idea by now) all the contacts in the database who have at least one Web Site.

Of course, I could go on forever (and I would if I were less ambitious as I could create a hundred page article by simply cutting and pasting the above query and simply changing the last line) but I think you get the point.

UNION:

Another, often overlooked SQL function is the UNION. Union brings two similar (exactly similar as you'll see) result sets together. An example may be the best way to describe this. One of my customers wanted to return all the contact records in the database. "Hey", you say, "You can do that easy with a contact listing". But, I correct you, the customer wants to see ALL the contact, including the secondary all bunched together in a big long list. Not so easy; in fact, there is not a way to do this in GoldMine except with a report which is useless if you want to do anything with the data.

Let's tackle this one step at a time. First, the primary contacts:

```
SELECT
        c1.company,
        c1.contact,
        c1.address1,
        c1.address2,
        c1.city,
        c1.state,
        c1.zip
FROM
        contact1 c1
```

Okay, easy enough. Now how about the secondary contacts:

```
Select * from contsupp where rectype=’C’
```

Right? Wrong! You end up with two different lists, which isn’t what the customer requested. Not to mention you don’t have the same information for each contact when you do it this way.

You could change query number two to this:

```
SELECT
        c1.company
        cs.contact
        cs.address1
        cs.address2
        cs.city
        cs.state
        cs.zip
        cs.accountno
FROM
        contact1 c1
INNER JOIN
        contsupp cs
ON
        c1.accountno=cs.accountno
WHERE
        rectype='C'
```

Now at least you have two similar result sets so you could do a cut and paste job in Excel or something, however, you’re creating a lot of extra work.

The UNION function combines two similar result sets into one big list.. hey, that sounds like what we’re looking for so let’s try it:

```
SELECT
        c1.company,


        c1.contact,
        c1.address1,
        c1.address2,
        c1.city,
        c1.state,
        c1.zip
FROM
        contact1 c1

UNION

SELECT
        c1.company
        cs.contact
        cs.address1
        cs.address2
        cs.city
        cs.state
        cs.zip
        cs.accountno
FROM
        contact1 c1
INNER JOIN
        contsupp cs
ON
        c1.accountno=cs.accountno
WHERE
        rectype='C'
```

Should work beautifully right? Wrong again. The UNION operator has one annoying (but perfectly logical) quirk in that the result sets to be combined must be identical in regards to the data types and sizes of the columns. If you look through GoldMine's Data dictionary, you'll notice that the data fields in contact1 are not necessarily the same size as the data fields in the contsupp table where similar data is stored (i.e. Address1), so you'll need to "CAST" each column in the second query to match the data types and sizes of the fields used in the first. Like so…

```
SELECT
        c1.company,
        c1.contact,
        c1.address1,
        c1.address2,
        c1.city,
        c1.state,
        c1.zip
FROM
        contact1 c1

UNION

SELECT
        CAST(c1.company AS char(40)),
        CAST(cs.contact AS char(40)),
        CAST(cs.address1 AS char(40)),
        CAST(cs.address2 AS char(40)),
        CAST(cs.city AS char(30)),
        CAST(cs.state AS char(20)),
        CAST(cs.zip AS char(10)),
        cs.accountno
FROM
        contact1 c1
INNER JOIN
        contsupp cs
ON
        c1.accountno=cs.accountno
WHERE
        rectype='C'
```

This works beautifully.

Another case where the UNION operator comes in handy is when you want to do a report that includes a series of record counts.

Let’s say, for instance, that you are storing a contact type value in the key1 field. Your contacts belong to any one of the following contact types:

Suspect
Prospect
Customer

In other words, your contact records have one of the above values in the key1 field. You would like to know what the record distribution is for your entire database. In other words you want to know how many Suspects you have vs. how many Prospects, vs. how many Customers and so on.

You probably already know how to do record counts, so you could simply write three separate queries and do a cut and past job to get the whole picture.   Something like this:

```
Select count(*) from contact1 where key1='Suspect'
```

Cut and Past into Excel

```
Select count(*) from contact1 where key1='Prospect'
```

Cut and Past into Excel

```
Select count(*) from contact1 where key1='Customer'
```

Cut and Past into Excel

You have a sudden brainstorm and translate this into

```
Select count(*) from contact1 where key1='Suspect'
UNION
Select count(*) from contact1 where key1='Prospect'
UNION
Select count(*) from contact1 where key1='Customer'
```

And now you have a result that looks like this:

```
Row     Count___
1       38
2       100
3       330
```

Pretty cool, saves a few cuts and pastes. But why not just finish the job:

```
Select count(*),(CAST('Total Records' as varchar(80))) from contact1
UNION
Select count(*),(CAST('Total Suspects' as varchar(80))) from contact1 where
key1='Suspect'
UNION
Select count(*),(CAST('Total Prospects' as varchar(80))) from contact1 where
key1='Prospect'
UNION
Select count(*), (CAST('Total Customers' as varchar(80))) from contact1 where
key1='Customer'
UNION
Select count(*), (CAST('Total Undefined' as varchar(80))) from contact1 where key1
not in ('Customer','Suspect','Prospect')
```

Which returns something like this:

```
Row     Count___        Total_reco
1       52      Total Customers
2       100     Total Suspects
3       330     Total Prospects
4       5554    Total Undefined
5       6036    Total Records
```

Which, of course eliminates cutting and pasting and allows you to save the report for someone else to run at their leisure without having to know all the dirty details of running three or four queries to get a single report.

Sub queries:

Another useful feature of SQL is the ability to use sub queries. In fact, we've already done this in our fist example (IN and NOT IN) when we limited our result sets like this:

```
WHERE
        Accountno
IN
        (Select accountno from contsupp where contact like 'Web Site%')
```

Sub queries can also be used to combine information that belongs in different data tables but doesn't fit together well with a plain old join.

On more than one occasion, I've been asked by customers to write a report in a wide line type format that includes the next calendar entry (next step) or the last history item as columns in the result set. Obviously there is no way to do this in GoldMine. You can print a history report and plenty of calendar reports, but nothing in wide line format that includes JUST the last history item or the next calendar item.

As you can probably guess, I'm about to show you how to do exactly that with an SQL Query. This, by the way, is an excellent example of how queries are handled differently on the SQL vs. the Dbase version of GoldMine**. The following example is NOT possible on the Dbase version** due to the lack of support for the "TOP" function (a horrible limitation and oversight on Borland's part in my opinion) in addition to the lack of support for calculated column names (as in *SELECT contact as 'Contact'*)

First, lets get our contact1 columns:

```
SELECT
        Accountno,
        Company,
        Contact,
        City,
        State,
        Phone1
FROM
        Contact1
```

No mystery here, however, here's the cool part. Let's add some sub queries.

```
SELECT
        C1.Accountno,
        C1.Company,
        C1.Contact,
        C1.City,
        C1.State,
        C1.Phone1,
        (SELECT TOP 1 ondate FROM conthist WHERE accountno=c1.accountno
        ORDER BY ondate DESC) as ‘Date’,
        (SELECT TOP 1 userid FROM conthist WHERE accountno=c1.accountno ORDER
        BY ondate DESC) as ‘User’,
        (SELECT TOP 1 ref FROM conthist WHERE accountno=c1.accountno ORDER BY
        ondate DESC) as ‘Reference’
FROM
        Contact1 as C1
```

Okay, a couple of comments; The key here is that by using column aliases (c1.xxxx) you’re able to refer to a column outside of your sub query:

(SELECT TOP 1 ondate FROM conthist WHERE accountno=**c1.accountno** ORDER BY ondate DESC) as ‘Date’

You’ll notice, as soon as you inspect the results that the query respects your current position In the Contact1 table. If you neglect to use table aliases, you either have to call the table by name (i.e. contact1.fieldname) or the Query interpreter will throw an error because it thinks you’re referring to the wrong table (in our case, the accountno in the contact history table instead of the contact1 table).

You also have to specify the sort order as “Descending” which is opposite of the default value of “Ascending”. If you neglect to do this, you’ll get the information from the oldest history item rather than the latest.

Now, let’s add the “Next action” to the query by doing the same thing but with the calendar file instead. Since this query won’t run on the Dbase version for other reasons, we don’t need to worry about specifying the physical file locations as mentioned above.

```
SELECT
        C1.Accountno,
        C1.Company,
        C1.Contact,
        C1.City,
        C1.State,
        C1.Phone1,
        (SELECT TOP 1 ondate FROM conthist WHERE accountno=c1.accountno
        ORDER BY ondate DESC) as ‘Date’,
        (SELECT TOP 1 userid FROM conthist WHERE accountno=c1.accountno ORDER
        BY ondate DESC) as ‘User’,
        (SELECT TOP 1 ref FROM conthist WHERE accountno=c1.accountno ORDER BY
        ondate DESC) as ‘Reference’,
        (SELECT TOP 1 ondate FROM cal where accountno=c1.accountno ORDER BY
        ondate DESC) as ‘Next Act Date’,
        (SELECT TOP 1 userid FROM cal where accountno=c1.accountno ORDER BY
        ondate DESC) as ‘Next Act User’,
        (SELECT TOP 1 rectype FROM cal where accountno=c1.accountno ORDER BY
        ondate DESC) as ‘Next Act Type’,
        (SELECT TOP 1 ref FROM cal where accountno=c1.accountno ORDER BY
        ondate DESC) as ‘Next Act Reference’
FROM
        Contact1 as C1
```

Which returns a very wide report which doesn’t fit nicely in this document but probably looks exactly like you’d expect it to look. You can cut and paste it into GoldMine’s SQL Query tool and see for yourself.

Wrap Up

I hope that, while we’ve only looked at a few examples, you’ll find a few ideas here that may give you some inspiration the next time you’re stuck. I hope to follow this article up with another installment as soon as I can find some time….

Russell Smallwood is CEO of Relatia Software Corporation, authors of the popular Relatia Time and Billing GoldMine add-on. Relatia also provides custom programming services to GoldMine VARs across the US.

For more information on how you can help your customer and win more business, call Linda O’Connor at (770) 663-4455 x 305 today.