# High Level Process for Improving the Efficiency of Discern Explorer Programs and Queries:

Here is the process I generally use when trying to help someone improve the performance of a program.

The first thing I do is a high level scan of the source code looking for programming practices that violate the efficiency guidelines.  Those guidelines are documented in the Discern Explorer Reference Pages under the All About Program Efficiency Checklist link : <https://wiki.ucern.com/display/reference/All+About+Discern+Explorer+Program+Efficiency+Checklist>. I strongly encourage everyone to study those guidelines until they have a good understanding of them. Just knowing the guidelines will go a long way in ensuring you write efficient programs.

Some of the most common issues I see during the high level scan are:

1. using Dummyt tables in the middle of Plan/Join clauses
2. using Dummyt table instead of Expand() to get information from RDBMS tables that is related to a value in a record structure
3. using unnecessary queries
4. hitting the code\_value table instead of using UARs
5. improper use of the alterlist() function

 If I see things that I think fall under items 1-3 listed above, I generally try to correct them first because they can dramatically affect the queries that are passed to the RDBMS.

 Next, if I have access to the environment where the program is actually being executed, I will open ExplorerAnalyzer.exe (EA).  EA is a front end application and you should be able to run it from the same place where you run your other Discern Explorer applications.  EA shows you information about the system resources the RDBMS used to execute queries.  EA opens to a SQL Area tab that allows you to search for the statistics for queries that are currently in the shared memory pool.  If you enter a program name and click the execute button, EA will display the statistics for all the queries for that program.  The top half of the screen will be the stats.  If you click one of those rows the lower half of the screen will show the query that was passed to the RDBMS that the stats are for.  That helps to identify which queries in the program are the worst performing ones.  Once I identify which queries perform the worst, I then turn to CCLQUERY to see how the RDBMS is processing those queries. (If I don't have access to an environment where the program was executed I just run CCLQUERY on the queries that I suspect are the worst performers.)

 Many performance problems are caused by the RDBMS optimizer choosing to use the wrong index, or the programmer not writing the queries in a way that allows using indexes.  CCLQUERY prompts for a program name.  It then pulls all the queries out of the program and asks the optimizer how it would process those queries.  The output shows the plan that the optimizer would use if it executed those queries.  CCLQUERY ignores all commands that are not queries. So if the program does something like:

If (condition = 1)

     Select one...

Else

     Select two...

Endif

 The IF statement is ignored and the plan for how the optimizer would process both Select one and Select two would be included in the output of CCLQUERY.  If a program has a lot of queries it can sometimes be difficult to understand which plan shown in CCLQUERY goes with which query in the Discern Explorer program.  Since you don't have to actually execute the query/program to use CCLQUERY on it, I will often create a stub program that just contains a single query.  Then run CCLQUERY on that stub program.  That way I can look at just the plan for that single query.  If the query uses the dummyt table in the middle of a plan/join that single Discern Explorer query will be processed as separate queries at the RDBMS level.  But my opinion is that it is often easier to work with just the plan from one specific Discern Explorer query than it is all the queries in a real program. (I have actually customized the Reports menu in DVDev to have an option that runs CCLQUERY and passes "MINE", "BGR\_TEST\_CCLQUERY", "Y" as parameters. Then when I want to see how the RDBMS would process a specific query, I copy/paste that query into a blank file, wrap drop/create program bgr\_test\_cclquery and end go around that query, and compile the file. I can then select my custom report from the menu to see the output of CCLQUERY for that specific query.)

So I use EA to help identify the worst performing queries in the program.  Then use CCLQUERY to see how the optimizer is processing those queries.  Most of the time a query that is not performing well can be improved by getting the optimizer to choose a different index than the one it is choosing.  We generally do that by using +0 or a function to prevent the optimizer from choosing the wrong index.  Sometimes we also have to add qualifications.  That part of it can become an art but the techniques are covered in detail in the Program Efficiency Checklist mentioned earlier.  It often requires trying several ways of structuring the query to find the best way.

# Let's look at a specific example:

A client posted the following example and asked for recommendations for improving its efficiency.

SELECT INTO "nl:"    
PATIENT=P.NAME\_FULL\_FORMATTED,  
FIN\_nbr= ea.alias,  
REASON\_FOR\_VISIT=E.REASON\_FOR\_VISIT,  
EVENT\_DT=C.EVENT\_END\_DT\_TM "@SHORTDATETIME",  
C.EVENT\_CD,  
EVENT = UAR\_GET\_CODE\_DISPLAY( C.EVENT\_CD ),  
info= c.event\_tag  
  
FROM  
CLINICAL\_EVENT  C,  
ENCOUNTER  E,  
PERSON  P,  
ENCNTR\_ALIAS  EA  
  
PLAN  C  
WHERE  C.clinsig\_updt\_dt\_tm between cnvtdatetime( cnvtdate($startdt ), 0 )  
AND  cnvtdatetime( cnvtdate( $enddt ), 235959 )  
and c.event\_cd in (48115255,68627772) ;Pneumonia Vaccine  
and c.valid\_until\_dt\_tm+0 > sysdate  
and c.view\_level = 1  
and not c.result\_status\_cd in (inerror\_cd,inerror2\_cd)  
  
JOIN e  
WHERE E.encntr\_id = C.encntr\_id  
AND C.person\_id+0 = E.person\_id  
and e.end\_effective\_dt\_tm > sysdate  
and e.active\_ind = 1  
  
JOIN ea  
WHERE ea.encntr\_id = outerjoin(e.encntr\_id)  
and ea.encntr\_alias\_type\_cd = outerjoin(141)  
and ea.end\_effective\_dt\_tm > outerjoin(sysdate)  
and ea.active\_ind = outerjoin(1)  
  
JOIN p  
WHERE p.person\_id = e.person\_id  
and p.end\_effective\_dt\_tm > sysdate  
and p.active\_ind = 1  
  
order by e.encntr\_id ;  
  
head report  
  ecnt = 0  
  
head e.encntr\_id  
  
    ecnt = ecnt + 1  
    stat = alterlist(encounters->qual,ecnt)  
    encounters->qual[ecnt].encntr\_id = e.encntr\_id  
    encounters->qual[ecnt].person\_id = p.person\_id  
    encounters->qual[ecnt].pat\_name = p.name\_full\_formatted  
    encounters->qual[ecnt].fin\_nbr = ea.alias  
    encounters->qual[ecnt].visit\_reason = reason\_for\_visit  
    encounters->qual[ecnt].info = substring(1,60,info)  
  
with nocounter  
  
;-------------------------------------------------------------  
; get date vaccine given  
select into "nl:"  
from  
  (dummyt d with seq=value(size(encounters->qual,5))),  
  clinical\_event ce,  
  ce\_date\_result cdr  
  
plan d  
join ce  
  where ce.person\_id = encounters->qual[d.seq].person\_id  
    and ce.event\_cd in (48115262)  
    and ce.encntr\_id = encounters->qual[d.seq].encntr\_id  
    and ce.publish\_flag = 1  
    and ce.valid\_until\_dt\_tm+0 = cnvtdatetime("31-DEC-2100,00:00:00")  
    and ce.record\_status\_cd = active\_cd  
  
join cdr  
where cdr.event\_id = outerjoin(ce.event\_id)  
  and cdr.valid\_until\_dt\_tm+0 > outerjoin(sysdate)  
  
order  
  d.seq,  
  ce.event\_id desc  
head d.seq  
  encounters->qual[d.seq].date\_given = format(cdr.result\_dt\_tm,"MM/DD/YY;;d")  
with nocounter  
;-------------------------------------------------------------  
; get vaccine detail  
select into "nl:"  
from  
  (dummyt d with seq=value(size(encounters->qual,5))),  
  clinical\_event ce  
  
plan d  
join ce  
  where ce.person\_id = encounters->qual[d.seq].person\_id  
    and ce.event\_cd in (24091791)  
    and ce.encntr\_id = encounters->qual[d.seq].encntr\_id  
    and ce.publish\_flag = 1  
    and ce.valid\_until\_dt\_tm+0 = cnvtdatetime("31-DEC-2100,00:00:00")  
    and ce.record\_status\_cd = active\_cd  
  
  
order  
  d.seq,  
  ce.event\_id desc  
head d.seq  
  encounters->qual[d.seq].what\_given = ce.event\_tag  
  encounters->qual[d.seq].event\_dt = format(ce.event\_end\_dt\_tm,"@SHORTDATETIME")  
with nocounter  
;-------------------------------------------------------------  
; get vaccine date  
select into "nl:"  
from  
  (dummyt d with seq=value(size(encounters->qual,5))),  
  clinical\_event ce  
  
plan d  
join ce  
  where ce.person\_id = encounters->qual[d.seq].person\_id  
    and ce.event\_cd in (20077166)  
    and ce.encntr\_id = encounters->qual[d.seq].encntr\_id  
    and ce.publish\_flag = 1  
    and ce.valid\_until\_dt\_tm+0 = cnvtdatetime("31-DEC-2100,00:00:00")  
    and ce.record\_status\_cd = active\_cd  
    AND NOT CE.event\_tag IN ("Date\Time Correction","In Error")  
  
  
order  
  d.seq,  
  ce.event\_id desc  
head d.seq  
    encounters->qual[d.seq].event2\_dt = ce.event\_tag ;event\_end\_dt\_tm,"@SHORTDATETIME")  
with nocounter

After reviewing the code using the high level process discussed above, I have the following questions:

What is the total number of clinical events that would have a clinsig\_updt\_dt\_tm that is between the dates entered for $startdt and $enddt and how long would it take to just run the first query?

If I run CCLQUERY on the first query in the program I get the following output:

6)TABLE ACCESS BY INDEX ROWID CLINICAL\_EVENT

7)INDEX RANGE SCAN XIE15CLINICAL\_EVENT NON-UNIQUE

>>> NONUNIQUE CLINSIG\_UPDT\_DT\_TM

6)TABLE ACCESS BY INDEX ROWID ENCOUNTER

7)INDEX UNIQUE SCAN XPKENCOUNTER UNIQUE

>>> UNIQUE ENCNTR\_ID

5)TABLE ACCESS BY INDEX ROWID PERSON

6)INDEX UNIQUE SCAN XPKPERSON UNIQUE

>>> UNIQUE PERSON\_ID

4)TABLE ACCESS BY INDEX ROWID ENCNTR\_ALIAS

5)INDEX RANGE SCAN XIE2ENCNTR\_ALIAS NON-UNIQUE

>>> NONUNIQUE ENCNTR\_ID

The above shows that the optimizer is reading the Clinical\_Event table using the index on clinsig\_updt\_dt\_tm. The clinical\_event table is huge. If you just run this query, without doing any of the record structure loading, how long does it take to complete? If it will complete in a reasonable amount of time it may be okay. But you may need to restructure this query to plan on one of the other tables using a different index. For example could you start on the Encounter table using the reg\_dt\_tm or other index?

My next question is do you really need the record structure? The first query gets some specific clinical\_events. The subsequent selects get additional clinical\_events for the same encounters. Could you get all of those clinical events using one read on the clinical\_event table if you added some OR qualifications? For example the first query uses:

PLAN C

WHERE C.clinsig\_updt\_dt\_tm between cnvtdatetime( cnvtdate($startdt ), 0 )

AND cnvtdatetime( cnvtdate( $enddt ), 235959 )

and c.event\_cd in (48115255,68627772) ;Pneumonia Vaccine

and c.valid\_until\_dt\_tm+0 > sysdate

and c.view\_level = 1

and not c.result\_status\_cd in (inerror\_cd,inerror2\_cd)

The second query uses

join ce

where ce.person\_id = encounters->qual[d.seq].person\_id

and ce.event\_cd in (48115262)

and ce.encntr\_id = encounters->qual[d.seq].encntr\_id

and ce.publish\_flag = 1

and ce.valid\_until\_dt\_tm+0 = cnvtdatetime("31-DEC-2100,00:00:00")

and ce.record\_status\_cd = active\_cd

Could you combine those into a single query that used something like:

PLAN C

WHERE C.clinsig\_updt\_dt\_tm between cnvtdatetime( cnvtdate($startdt ), 0 )

AND cnvtdatetime( cnvtdate( $enddt ), 235959 )

and (c.event\_cd in (48115255,68627772) ;Pneumonia Vaccine

and c.valid\_until\_dt\_tm+0 > sysdate

and c.view\_level = 1

and not c.result\_status\_cd in (inerror\_cd,inerror2\_cd))

OR( ce.event\_cd in (48115262)

and ce.publish\_flag = 1

and ce.valid\_until\_dt\_tm+0 = cnvtdatetime("31-DEC-2100,00:00:00")

and ce.record\_status\_cd = active\_cd)

It is possible that all the record structure processing could be eliminated using additional OR qualifications on the clinical\_event table in the first query. If it's not necessary, eliminating all the record structure processing could save you quite a bit of time.

Assuming there are reasons for using the record structure I would make the following recommendations:

head e.encntr\_id

    ecnt = ecnt + 1  
    stat = alterlist(encounters->qual,ecnt)

/\* How many encounters are likely to qualify.  If it will generally be more than two, instead of using alterlist to allocate memory on every encounter, add code to allocate memory in reasonable sized blocks.\*/  
    encounters->qual[ecnt].encntr\_id = e.encntr\_id  
    encounters->qual[ecnt].person\_id = p.person\_id  
    encounters->qual[ecnt].pat\_name = p.name\_full\_formatted  
    encounters->qual[ecnt].fin\_nbr = ea.alias  
    encounters->qual[ecnt].visit\_reason = reason\_for\_visit  
    encounters->qual[ecnt].info = substring(1,60,info)

Instead of using the dummyt table in the queries that

     ; get date vaccine given

     ; get vaccine detail

     ; get vaccine date

use the Expand() function.

 When you use the dummyt method it causes a select to be executed at the RDBMS level one time for each item in the record structure list. For example in your second select you have the following:

select into "nl:"

from

(dummyt d with seq=value(size(encounters->qual,5))),

clinical\_event ce,

ce\_date\_result cdr

plan d

join ce

where ce.person\_id = encounters->qual[d.seq].person\_id

and ce.event\_cd in (48115262)

and ce.encntr\_id = encounters->qual[d.seq].encntr\_id

…

The above would cause a query something like the following to be passed to the RDBMS

select…

from clinical\_event ce

where ce.person\_id = :1 and ce.event\_cd = 48115262

and ce.encntr\_id = :2

…

The above is going to be executed one time for each position you have the encounter->qual list. If there are 10 encounters that select will be executed 10 times. If there are 100 encounters that select will be executed 100 times. Each time the query is executed :1 and :2 will be set to one of the values from the record structure list. Executing that query multiple times, getting the data and holding it in a temporary buffer and then putting it all together to create a final result set is very inefficient. So the above should probably be changed to something like:

select into "nl:"

from

clinical\_event ce,

ce\_date\_result cdr

plan ce

where expand(num,1, size(encounters->qual,5),

ce.person\_id, encounters->qual[num].person\_id),

ce.encntr\_id, encounters->qual[num].encntr\_id)

and ce.event\_cd in (48115262)

…

Using the above only one query is going to be passed to the RDBMS and it will most likely be much more efficient.