

Re: Opinions on approach, please...

I think it could be achieved with:

```
EXEC SQL  
DECLARE cursor cur01 for (SELECT * from Atable  
WHERE KeyField <=  
someValue ORDER BY KeyField DESCENDING)  
END-EXEC
```

...but I'm certainly not sure, and things are difficult enough
without
that particular added complication :-).

I haven't used START against an indexed file in a batch COBOL program

very

often (on IBM mainframe it would be KSDS VSAM of course). In CICS it
would be handled with EXEC CICS START BROWSE, READ NEXT (or
PREV), END
BROWSE. If I recall correctly, START doesn't really do anything except
point to some position in the file for subsequent sequential processing.

The real action is in the READ NEXT or READ PREVIOUS.

Yes, that's correct. Normally there would be a series of READs following
the
START. It's called "skip sequential" processing.

The process continues until either another START is encountered or EOF.

To emulate this I plan on using an SQL cursor. It would be defined and
opened by a START, following READs get translated to SQL FETCHes, and if
we
hit EOF or another START is received, we CLOSE the cursor.

This is the bit I'm not too sure about at the moment. I could do it with

dynamic SQL, or I can have prebuilt static code snippets I can insert
into
the application. (This is possible because all fields are always
returned,
so the only "variable" part of the cursor is the condition specified in
the
START. I can parse this and pass it as a relational operator and a value
to

Re: Opinions on approach, please...

Re: Opinions on approach, please...

the MOST COM server, or I can pass it as a COBOL string and use dynamic SQL to include it into the WHERE clause of a dynamic cursor within the MOST server There are pros and cons and I keep spinning between them... :-)

Is this the kind of thing you are thinking about?

```
exec sql
declare ft_select cursor for
select BRCH_NBR, ACCT_NBR, POST_DATE, AMOUNT,
SERIAL_NBR, SEQUENCE_NBR, POST_FLAG
from FILM.FILM_TRANSACTIONS
where BRCH_NBR = :ft-brch-nbr
and ACCT_NBR = :ft-acct-nbr
and (
AMOUNT = :FT-AMOUNT
OR
AMOUNT = :FT-AMOUNT-NEG
or
:amount-flag = 'N'
)
and (
SERIAL_NBR = :ft-serial-nbr
or
:serial-flag = 'N'
)
end-exec
```

Essentially, the above is so I can search on:

- amount only (any serial number) matches (set serial-flag to 'N')
- serial number only (any amount) matches (set amount-flag to 'N')
- both amount and serial number match (neither flag is 'N')

Its rather ugly, but it works (or it did the last time I checked it; not production code).

That would establish a position OK, but it means nothing. We need to fetch sequentially from that point. That's why a cursor looks like being necessary. I COULD do it with a resultset if it wasn't in COBOL or could

even use LINQ which would generate a Lambda for it, but that is all off the table because of the environment it has to run in (Workstation COBOL). A

cursor is MUCH less efficient than a resultset or LINQ, but it is the best solution I can see wioth ESQL in COBOL.

Re: Opinions on approach, please...

Re: Opinions on approach, please...

Every time you say this I have to wonder why a 'result-set' is so much more efficient. Is it because when you FETCH from a cursor you go back to the database each time to get the next record, whereas with a result set the database returns the entire result immediately?

DB2, at least, offers a "FOR READ ONLY" option which appears to do 'result blocking', in that (I think!) it returns blocks of multiple rows, so your fetch just goes against rows that the client already has in memory. Or something like that.

Here's some information from

<http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=/com.ibm.db2.udb.admin.doc/doc/r0000937.htm>:

"read-only-clause

 -FOR--+--READ--+--ONLY-----><

'-FETCH-'

The FOR READ ONLY clause indicates that the result table is read-only and therefore the cursor cannot be referred to in Positioned UPDATE and DELETE statements. FOR FETCH ONLY has the same meaning.

Some result tables are read-only by nature. (For example, a table based on a read-only view.) FOR READ ONLY can still be specified for such tables, but the specification has no effect.

For result tables in which updates and deletes are allowed, specifying FOR READ ONLY (or FOR FETCH ONLY) can possibly improve the performance of FETCH operations by allowing the database manager to do blocking. For example, in programs that contain dynamic SQL statements without the FOR READ ONLY or ORDER BY clause, the database manager might open cursors as if the FOR UPDATE clause were specified. It is recommended, therefore, that the FOR READ ONLY clause be used to improve performance, except in cases where queries will be used in positioned UPDATE or DELETE statements.

A read-only result table must not be referred to in a Positioned UPDATE or DELETE statement, whether it is read-only by nature or specified as FOR READ ONLY (FOR FETCH ONLY)."

I just did a test and even without specifying FOR READ ONLY I was able to get all of the results for a particular query in a single block. I know this because I used Wireshark to trace the TCP/IP packets and I could set it.

When I added a "FOR UPDATE" clause I ended up getting one at a time.

Re: Opinions on approach, please...

Re: Opinions on approach, please...

Anyway, not sure if this is the performance issue, but its certainly worth knowing.

Of course I also have no idea what database you are using either. This is DB2/LUW 9.5 using the DB2 supplied pre-compiler.

Frank

.