Base One International Corporation
44 East 12th Street
New York, NY 10003
212-673-2544
info@boic.com
www.boic.com

Dr. GUI is NOT Dr. Database

- How Base One's Scroll Cache simplifies database browsing & improves performance

MSDN News magazine from Microsoft regularly features a column called "Ask Dr.GUI" where answers are given for various technical questions. The column is usually good, very informative and useful for the developer community.

However in the September/October 2001 issue of MSDN News in the column "Ask Dr.GUI #60" there is a simple question regarding the best way to browse through an SQL result set. Dr. GUI's proposed solution is both inefficient and impractical!

The question posed is important and a common one that is relevant to most database applications and illustrates why Base One's Scroll Cache database class is needed. Here's the extract from Dr. GUI's column in MSDN (Microsoft Developer Network):

> **Jumping Records**
> **Dear Dr. GUI:**
> I am developing a Web application using Active Server Pages (ASP). I want to display a list of items that come from a database and I need to divide the list into pages. For each page, the user requests that I open an ADO recordset, divide it into pages, and send back the appropriate page. The problem in this process is that for each page view, my database (SQL Server) has to return a recordset with all records and the Internet Information Services (IIS) displays one page out of it. I was wondering if there was a better way to make SQL Server only return the records I need. I know that to get the first 25 records (page #1), I can use SELECT TOP 25, but how can I get records 26-50 (page #2)? Do I need to do SELECT TOP 50 and jump to record #26?
>
> Dan Avni
>
> **Dr. GUI replies:**
> All this jumping back and forth feels like a game of hopscotch to the good doctor. Let's get back to having both feet planted firmly on the ground and answer your question.
> You can certainly use the SELECT TOP 25 FROM Table for the first page of 25 rows. For the next 25 rows, if you query the database again and jump to record #26, the data might have changed by that time and record #26 may no longer be the appropriate row. Further, if you do not use the

ORDER BY clause in the select statements, then the order in which the rows are listed might not be consistent if there is no clustered index on the table. Ideally, SQL Server should return the entire recordset and the selection of pages should be handled at the ADO level. If the number of rows being returned to IIS is very big, then you can use a Select Into statement to store all the rows from the initial Select statement in a temporary table. Then you can use a cursor to browse through the temporary table, in chunks of 25 rows at a time. Please note: it is important that you properly manage transactions (preferably not using a transaction in this case) and also the connection to SQL Server. Otherwise the preceding solution could lead to blocking of resources.

Flaws in the solution suggested by Dr. GUI.

"You can certainly use the SELECT TOP 25 FROM TABLE …"

This SQL query will work only with Microsoft SQL Server and not with other database systems because TOP 25 is nonstandard SQL.

"Ideally, SQL Server should return the entire recordset…"

What if the entire result set consists of hundreds of thousands of records? The more users the worse this "solution" becomes. The system would crawl. If the entire result set is returned just to display a few pages every application would crawl because of network and server overload resulting from repeatedly fetching and transferring large amounts of unneeded data.

"You can use a Select Into statement…"

SELECT INTO needs an output table name, which is the start of a complex mess. What table name should be used? If a table name is programmatically created, who drops the table? If the table gets dropped after the browsing is finished, what happens if the program exits abnormally and the table does not get dropped? Using "temporary" tables leads to complexity and nonstandard SQL. Once again, the more users, the worse it gets, since each user might need to create a table using SELECT INTO. That means estimating a potentially large, disk space requirements and coding for out-of-space conditions. If the same table is shared, all users would be required to compromise on the timeliness of the data and might have to accept old data created by the very first user. You get the idea. Using SELECT INTO creates extra programming work and adds needless database overhead and disk requirements – when all you want to do is scroll around and browse data.

"Please note: it is important that you properly manage transactions…"
"… (preferably not using a transaction in this case)"

Dr.GUI warns that programmers must use great care in transaction processing settings to avoid locking up the database. Transaction processing exists to protect the integrity, reliability, and recoverability of many database applications, most of which also need to incorporate a browsing feature. Certainly, Dr. GUI can't mean that mixing transaction processing and browsing in a single application is prohibited?

Wouldn't it be better if the transaction processing settings for browsing could be taken care of automatically? Browsing though a result set usually requires a standard set of scrolling operations, such as: moving page by page, going to the last page, and skipping to a page containing a particular value. In the GUI world, users love to browse with table controls (grids) and scroll bars. In web applications, users should be able to scroll using hyperlinks, such as Next Page, Previous Page, First Page and Last Page.

In Dr.GUI's scheme, all database applications that require transaction processing also require expert custom programming to prevent slow speed and occasional deadlocks. Programmers might get to discover performance problems and bugs related to transaction processing only late in the testing process. Even worse, it may be the users who discover these problems in production. Not so with Base One's Scroll Cache components.

Scroll Cache and the ideal way to implement browsing.

Now, let us examine features users need for browsing through a result set page by page:

- Next Page / Previous Page feature should be present. When Next Page is clicked the next page of data in the logical order to the current page should get displayed. Similarly the Previous Page also should function.

- First Page / Last Page feature should be present. If Last Page is chosen then the last page of the result set should be displayed. At any point if the First Page is chosen then the very first page of the result set should be displayed.

- Next Record / Previous Record or Next Line / Previous Line feature should be present. That means the current page of data should scroll up or down by one row.

- At any given time, all the rows on the whole page of data currently displayed should be in sync, a valid snapshot of the database at an instant in time. In addition, it should be easy for programmers to guarantee each page shows the most current data. That means for example if Next Page is chosen and if no action has been done on the database then the next logical page of data in a given order should get displayed. If some other users have added new data or have deleted or changed any existing data then that change should get reflected as the Next Page of data is fetched. Same thing applies all other browsing operations such as Previous Page, First Page, Last Page, Next Line, Previous Line, etc.

- It should be easy for programmers to insure that no data is stored locally or in a temporary table. Proper, current data pages should be easy to obtain afresh from the database. For example if the same current page is refreshed and if no changes have occurred in the database then the same page gets displayed. If any changes such as addition of new records that belong to the current page or change or deletion of records belonging to current page have taken place then the refresh operation should reflect those changes.

- The programmer should be able to easily control exactly the number of rows to be fetched during display processing. No unnecessary rows should be fetched or transmitted.

- Page fetching operation whether it is first page, last page, previous or next page it should take constant time. For example fetching the last page need not have to take a long time even if the result set consists of millions of records.

- No locks should be held or no resources sho uld get blocked waiting on some user action while the browsing is in progress. Immediately after the desired page of data (not the entire result set) is obtained from the database, all resources should be released.

- Browsing operation and all other database operations should take place under a valid transaction mode. Any dynamic changes that are made to the database by all the users using the same database should be visible whenever a new data page is obtained or when the current page is refreshed. Once a page is displayed, the user should be allowed to change or delete the records from the displayed page. In the meantime if any other user modifies the same record, then the user should get notified of the change. After seeing the modification done by some other user, the user should be allowed to retry the change or delete operation. Whenever a user changes or deletes a record or set of records in the current page and the transaction is committed, the current page should reflect those change when it gets refreshed. In the same way if any new records are added that belong to the current page, they should appear when the current page gets refreshed.

All of these features are completely provided by the Scroll Cache Record Set class of Base One's Foundation Class Library (BFC). Scroll Cache is a C++ class in the Database library that can be used by any Visual C++ application that interacts with IBM, Microsoft, Oracle and Sybase database systems. Scroll Cache is also available with COM and .NET interfaces, with C++, C#, VB, VB.NET, ASP, and ASP.NET samples showing usage of this database component.

BFC provides efficient, elegant and easy-to-implement solutions that address the complete range of database related issues. Scroll Cache is just one example of the components that the BFC toolkit provides for all sorts of database applications, big or small, web application or client server, 2-tier or 3-tier.

Bottom line: Scroll Cache speeds up queries and simplifies programming. It greatly reduces the load on the database server by freeing resources after every database operation and by insuring indexes are used - to avoid sorts. The result is greatly increased scalability – far more simultaneous end-users without degrading performance.