Optimising Client Side Data Entry Process

Nkechi ACHARA[1], BSc. and Nnorom ACHARA[2]*, PhD.

[1]Credit Suisse, Fixed income 2L Support, KVFS62, Uetlibergstrasse 231, 8070 Zurich
[2]MDPGA, Ministry of Defence, Wethersfield, Braintree, United Kingdom
* nnoromeluwa@btinternet.com

Abstract: Loss of data during data entry either by crashing or freezing is a common problem encountered in database data insert especially when there are many rows and columns of data to deal with and also when it is not appropriate to bind the data directly to the database. Techniques used in inserting data to the database include bulk load, row by row looping and the use of dataset together with data adapter. The dataset and looping techniques have been studied by comparing the time for data insertion. From the results it is concluded that the dataset technique should be used when the number of rows and columns grow beyond a threshold and below this threshold the row by row data insertion method marginally performs better.
Journal of American Science 2011;7(7):440-444]. (ISSN: 1545-1003). http://www.americanscience.org.

Key Words: Dataset, Data adapter, AJAX, OPENXML, Bulk load, Constraint, Data integrity, Primary key.

1. Introduction

Many data entry personnel may have encountered data loss after clicking to save the entry. These encounters occur most often in an environment such as creating duty roster and other places where, as a result of data validation, the best practice dictates that all entries must be made before saving to the database. As the data rows grow in number, clicking the save button may frustratingly result in the application crashing by "hanging" instead of saving to the database. Whenever a situation like this arises, the data entry personnel may resort to "killing" the session by resetting thus losing the work done so far or wait for the system to resolve the problem at its own choosing. The latter option is extremely a time consuming exercise. Resetting and losing all the data may be acceptable where only few records are involved but not where the lines run in hundreds. This situation is relevant in applications where the data at entry is not directly bound to the database. There are situations where it may not be advisable to directly bind the data to the database and these include where data validation is top of the priority list and roster systems. Depending on the circumstance, the integrity of data may be compromised if the data is directly bound to the database and so open the database to corruption. In roster systems, before saving to the database, the roster manager needs to have full visibility of all data relating to the duty date so as to enable him juggle and reconcile staff qualifications, skills and experience to match the tasks available for the day.

The majority of work on database performance is on the server side, hardly can one find any work on the client side where it is possible to lose hours of work if the system failed to save when required. Some of the approaches to optimisation of the server side include normalisation with the aim of keeping table indexing to a minimum, use of cursor and to a lesser extent, set theory. Even with the arrival of inexpensive storage devices, organisations would still have to guard that data does not grow in an exponential manner and also ensure that there is no loss in database performance. Others commission optimisation studies in order to have that extra edge over competitors.

Literature on client side performance is sparse. The few studies (Shah, 2007) on client performance focus their measurements on such issues as reducing the number of web pages by combining HTML pages with CSS files. Another technique found in the literature is the use of asynchronous processing to reduce full page reload and round trips to the database. In an asynchronous application response time is enhanced by the use of partial page reload which is essentially what happens with AJAX web application. Also found in the literature on client side performance is measurement studies where the focus is on memory usage reduction. (Koechley, 2007) discusses ways of improving web page retrieval including the removal of duplicate scripts and keeping AJAX cacheable. (Theurer, 2007 and 2011) investigated in experiments to find that it takes more than double the time to load all the cookie components on first visit to the server from the client side compared with subsequent visits. In another experiment Theurer showed that the time for cookie retrieval from the server to a page on the client side

depends on the size of the page. The study found that retrieval time doubled when the page size changed from empty to 3kb. In his own study for pages with many small objects, (Hopkins, 2011) investigated how HTTP client site implementation affects page load time. But for (Achara, 2009) where results varied widely and there was no much effort to handle the resulting spikes, there is virtually no studies found on ways to handle data insertion to the database as the size of data grows.

2. Data Entry Methods:

There are various techniques of inserting data to the database but it is useful to know which method is most appropriate for use in each given situation. With the right method chosen, the database data entry process will benefit and the data insertion time optimised. Some of the methods used to insert data into a database include: batch insert, bulk load, traditional row-by-row looping and the use of the in memory dataset together with the data adapter.

2.1 XML Bulk Load:

XML Bulk Load is a stand-alone COM object that allows the user to load semi-structured XML data into database tables. XML data can be inserted into a database by using the INSERT statement and the OPENXML function. The Bulk Load Utility provides better performance when inserting large amount of data. Because the XML source document can be large, the entire document is not read into memory for bulk load processing. Instead, XML Bulk Load interprets the XML data as a stream. As the utility reads the data, it identifies the database table(s), generates the appropriate record(s) from the XML data source, and then sends the records, in the case of MS database, to the SQL Server for insertion (MS Library, 2010). XML Bulk Load can operate in either a transacted or a non-transacted mode. Performance is usually optimal when bulk loading in a non-transacted mode by setting the transaction property to false and either the tables to be bulk loaded are empty or have unique indexes. Once in the database, it is memory intensive if the requirement, as in this study, is to convert the XML dump data to appropriate data type.

Oracle has also implemented bulk load utilities as a UNIX-level command which is issued directly from the UNIX shell (Oracle, 2010).

2.2 Dataset:

The dataset is an in-memory cache of data retrieved from a data source for example database (MS Library, 2011). The dataset is an important component of the ADO.NET architecture. The dataset may also be considered as a collection of data-table objects that use the data-relation objects to interact with one another. Data integrity can be enforced on the dataset using the unique constraint, primary and foreign key objects. Navigation between the data-tables in the dataset is achieved through data-relation objects. The dataset can read and write data schema as XML documents. The data and schema can then be transported across
HTTP and used by applications on any platform that is XML enabled.

The dataset has the ability to differentiate between Insert, Delete and Update queries. It uses the row-state property to decide what action to take. The row-state can be “Deleted”, “Modified”, “New” and “Unchanged”. The data adapter is the bridge between the dataset and relevant table in the database. For each query to work, there must be a primary key defined. Each data-table in the dataset should come from one database table. If there is the need to return or save data from or to different database tables, then a data-table collection with each data-table object created from each of the relevant database tables may be used. The dataset can be manipulated using the data-column, the data-row, constraint and data-relation.

2.3 Looping Method:

In the looping process each row of the record is dealt with one after the other and it is the most common and traditional method of inserting and saving data originating from tabular controls to the database. This technique is known to lead to performance drop as the number of rows on the table grows. Some application developers resort to batch operation by looping through a smaller number of rows in each insertion round. Under this walk around, the application may not crash but could lead to more-complex coding for the developer and overall, longer processing time since the cumulative time will be the sum of all the time for each server trip, the time taken to reposition the cursor after each batch operation and the associated time overhead. Some application developers favour the row by row looping method arguing that it gives the developer more scope to control the data insertion process.

The purpose of this study is to compare the data insertion time to the database between the looping method and the in memory dataset particularly as the number of rows and columns in the database table grow and recommend which method to adopt in any given situation.

3. Resource Requirement and Method

The laptop used in this study has the following features:

a. CPU: Intel (R) Core(TM)2 Duo P8600 @ 2.4GHz
b. RAM: 4GB
c. Hard drive: 283GB with 184GB free space.
d. OS: Windows Vista 32-Bit

Test database: SQL Server 2005 Express relational database.

A test database was designed and built using MS SQLServer 2005 Express running on a standalone development laptop

For a given number of rows and columns the times for data insertion in the database using the looping and dataset techniques were measured and recorded. The influence of rows was studied by fixing the number of database fields (columns) and varying the row number. Similarly, the effect of columns was studied by keeping the row constant and putting the variation on the number of fields.

The application is web based and runs on Microsoft .NET2 framework and the code is written in VB/C##. In each run, the application records and displays on the screen the times just immediately before and after the data insert as well as the time difference. Even though in real life the user will enter the data on the key board, for the purpose of this study, to save time on typing, the data was code generated. This does not however in any way affect the results.

4. Result and Discussion

In the course of running the test, it was observed that there was a tendency for results obtained from runs conducted using the same number of rows and columns to produce spikes and thus vary widely. It was therefore decided, for a given number of rows and columns, to run the application for at least eight times and average the values. Consequently, the time plotted for each row in figures 1 to 4 below is such averaged values.

Figure.1 is the result of data insertion to the database using the looping technique plotted on time against number of rows for given number of columns (fields) in the database table.

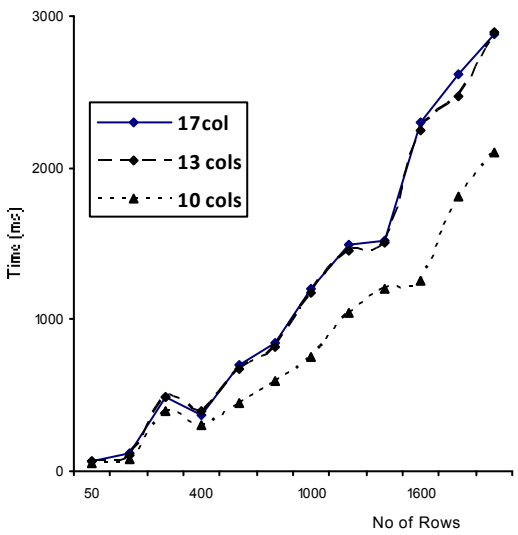

Figure.1 Time for Data Insert by Looping

Figure.2 is the result of data insertion to the database using the dataset technique plotted on time against number of rows for given number of columns (fields) in the database table.

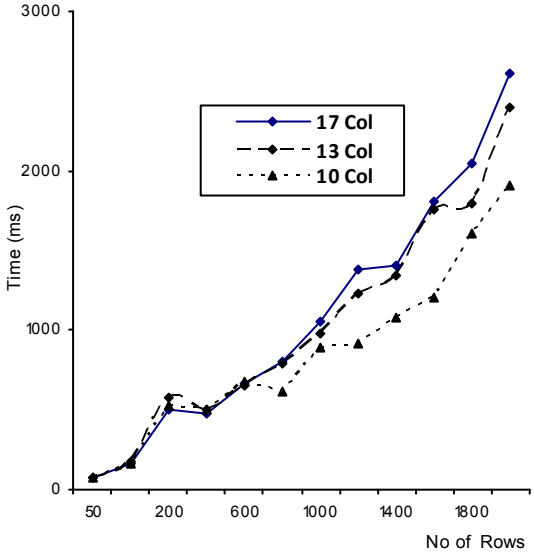

Figure.2 Time for Data Insert by Dataset

Figure.3 is the result of data insertion time comparison between the looping and the dataset methods for a database table having 10 columns.

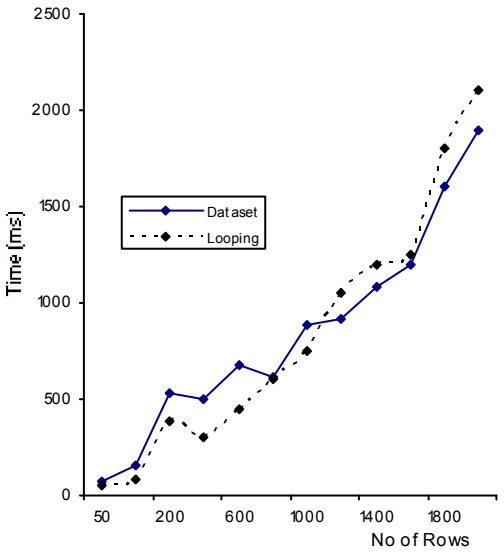

Figure.3 Dataset and Looping Comparison, 10 Columns

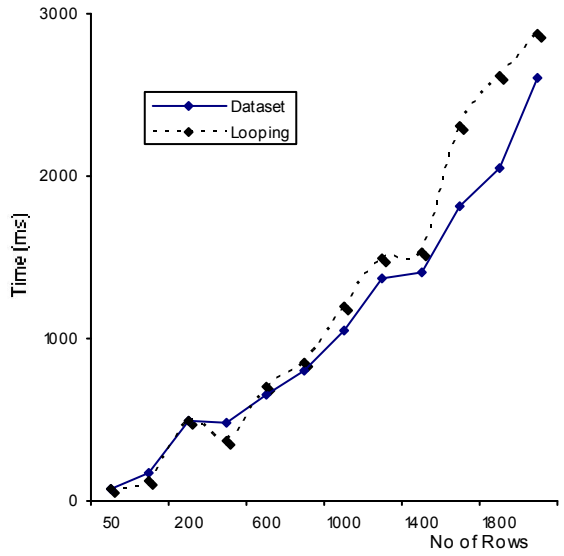

Figure.4 Dataset and Looping Comparison, 17 Columns

Figure. 4 is the result of data insertion time comparison between the looping and the dataset methods for a database table having 17 columns.

In figure.1 and figure.2, the time for insertion for any number of rows increases with increasing number of columns in the database table. Similarly for a given column in these two figures, the time for insertion increases with increasing number of rows. In figure.1 specifically, the time for data insertion between the plots for database table of 13 columns and a table of 17 columns can hardly be distinguished but the 13 columns plot is still marginally below the 17 columns plot. The time to insert data to the database in the 10 columns plot is however well below the other columns. In figure.2, below 400 rows the time for insertion for the database table of 17 columns plot falls below those of the table with 13 columns and 10 columns plots. This is not the trend that would be expected but it is thought that this is caused by the fixed overhead in using the dataset technique. This fixed overhead, it would appear, determines the time taken in saving the data when the row number is low. The result also appears to suggest that the time is independent of column number within this region.

In figure.3 and fig.4 the looping data insertion method to the database is compared with the dataset technique for given number of columns. On the whole, in each of the two figures, the time for insertion increases with increasing number of rows. In figure.3 which is plotted for a database table with 10 columns, below 1100 rows the dataset method performance is worse than the looping method. This may be attributed to the fixed overhead which is independent of any of the variables involved. Still on figure.3, above 1100 rows, the dataset data insertion method out performs the looping technique. In figure.4 which is plotted for a database table with 17 columns, below 500 rows, the looping method performs better than the dataset data insertion method. The fixed overhead may also offer an explanation for this behaviour. Above 500 rows in figure.4, the dataset database data insertion method once more performs better than its looping counterpart.

5, Conclusion:

1. Two database data insertion methods, the looping and the dataset have been compared.
2. For a given number of database table columns, below a threshold of rows, the looping method marginally performs better than the dataset technique.
3. For a given number of table columns, above a threshold of rows, the dataset database data insertion method out performs its looping counterpart. This is the region in the data entry system where data entry personnel encounter data loss due to crashing.
4. The experiments have shown that the dataset method performs better as the number of rows as well as columns grows and therefore under conditions

similar to this study, the dataset method should be employed when dealing with high number of rows and table columns.

6. References:

1. Shah, A., Client Side Web Site Performance, Retrieved from Onenaught.com, August 2007.
2. Koechley N, High Performance Web Sites, @Media Conference, London, 2007
3. Tenni Theurer, Perfornance Research Part 2: The Cache Browser – Exposed, www.onenaught.com, 2007.
4. Tenni Theurer, Perfornance Research Part 3: The Cache Browser – Exposed, www.onenaught.com, Retrieved 2011.
5. Aaron Hopkins, Optimising Page Load Time, www.die.net/musings/page_load_time, Retrieved 2011.
6. Nkechi Achara, Project for Partial Fulfilment for the Award of the Degree of BSc Computer Science, University of Hertfordshire, Hartford, 2009.
7. Microsoft Library, Introduction to XSD Schemas (SQLXML 4.0), Retrieved from www.msdn.microsoft.com/en-us/library, 2010.
8. Oracle Bulk Loader Overview, Retrieved from http://infolab.stanford.edu/~ullman/fcdb/oracle/or-load.html#overview#overview, Retrieved 2010.
9. MS Library, Retrieved from http://msdn.microsoft.com/en-us/library/system.data.dataset(v=vs.71).aspx, 2011.

5/29/2011