Published by :
http://www.ijert.org

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 6 Issue 04, April-2017

Pattern for Using SQL CE Database in Multithreaded Mode Along with C# .NET

Vu Van Minh Quang
Information Department
Ho Chi Minh City University of Foreign Language and Information Technology
City, Country: Ho Chi Minh City, Vietnam

***Abstract*— This paper presents a sample – Pattern is used to work with SQL CE in multithreaded mode. SQL CE is designed for single user and single application operations so although Microsoft has provided a SQL CE engine which has run quite well, Microsoft has not had best practices yet about how to effectively operate in the multithread environment with SQL CE. Along with illustrating the pattern, this paper also presents the testing result in the enterprise environment.**

I. INTRODUCTION

Microsoft SQL Server Compact

SQL CE (Compact) is a small relation database running without needing of a server. A software which use the SQL CE only needs some dll files (SQL CE's engine) provided by Microsoft to create and access SQL CE database files. Thus, the software could operate its database independently. It allows the database to work freely from installing a SQL server such as Microsoft SQL Server, MySQL or Oracle and so on.

Application's Scale: An SQL CE file has a maximum of 4GB and a software is able to create unlimited of this kind of file. We can consider the SQL CE is a shorten version of SQL Express, which just keeps essential functions of relation database. Therefore, the scale of software using SQL CE is also smaller. Microsoft designed SQL CE for apps running on Window Phone, Desktop Application and some apps on small websites.

Microsoft maybe designed SQLCE for focusing on software that have low concurrent users at a point of time when skipping the capability of concurrency of database. SQL CE allows many concurrent read to get access at a point of time, but such accessibilities must belong to a process (single process) and only open 256 concurrent connections.

The potential: with the capability of extending file up to 4GB, data which is saved on SQL CE is not small. In addition, SQL CE is provided completely free and it does not need to install more from the aspect of users. Along with the development of hardware, the capability of processing multithread, parallel process which is supported from .NET framework, the biggest problem of SQL CE is only how to manage connection more effectively.

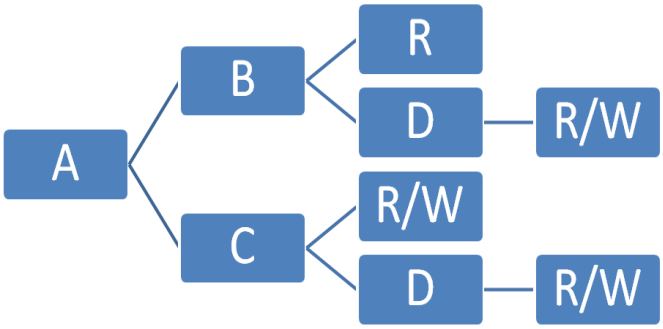
II. PROBLEMS AND ANALYSIS

To solve the problem of using SQL CE in the mode of multithread, the author will illustrate a practical case which applies a software for Company Z where has been running in food production area. The company needs to perform a statistical report about the business situation relating to retail activities. A Task Y, it processes single thread, in report it needs to process over the average level ranging from 100,000 to 200,000 lines of retail bill data. The average time for processing ranges from 400 to 500 lines per minute, total timing needs for accomplishing the report is approximately 4 – 6 hours. Hence, the problem concerned is how to curtail timing process.

Because app running under single thread, when it runs, total time of using a core CPU is accounted for 40 – 60 %. However, in terms of a multi-core CPU, the rest of core is wasted, in this case multi-core CPU merely uses one-sixteenth of core which means that over 90% of the strength of system is not exploited. Thus, one way to increase the process speed, it needs to consider multithread software in order to exploit potentially unused core.

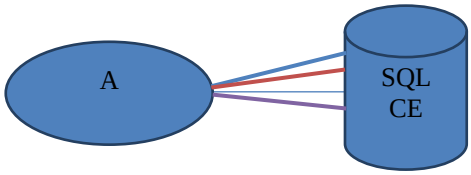
Task Description: for each line of retail bill, Y will carry out Task A.

Task A contains sub – functions B, C and D and import export manipulation on database as following the graph:



R: Read – Read data from database
R/W: Read/Write – Read and Write data on database

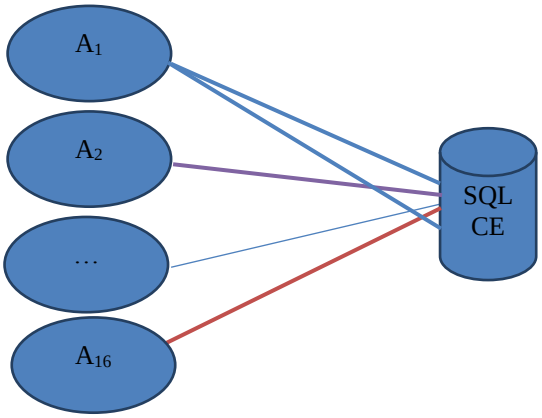
Look at the connection points database, Task A will have relatively dense relationships with database.



If Task A is implemented parallel with each retail bill, each task will implement one core, at the same time, it can be implemented 16 tasks. In this case, if it implements successfully, it can curtail the timing process 16 times. However, in this case, as SQL CE cannot implement parallel

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 6 Issue 04, April-2017

with manipulation of input data, it occurs the problem of synchronization with dense connection quantities.

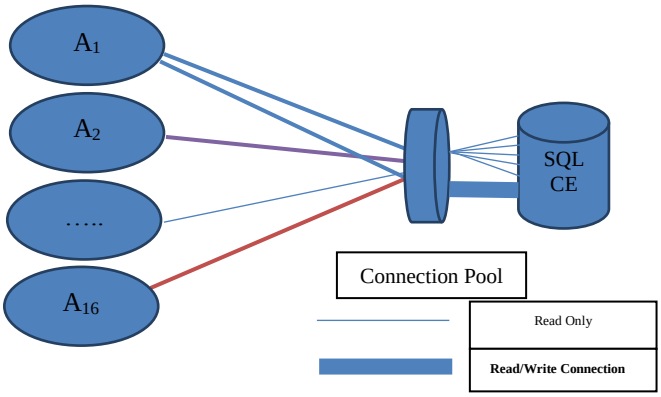


At a point of time, the number of Reading / Writing connections is 3*16 = 48 and the number of read-only connections is 1*16 = 16, the sum is 64 connections at a point of time leading to the occurrence of access failure on database file. Moreover, connection expense is relatively expensive (Using profiler for measure, creating expense and close connections accounts for 50% of the quantity of execution job).

III. SOLUTION

Need to have an intermediary layer so as to manage connections. The main mission of this layer is to solve two problems:

- To avoid creating and closing connections regularly: the use of connection pool to cache created connections instead of canceling and creating new.
- To manage Read / Write connections which can open many Read / Write connections at a point of time but is only allowed to open one Write connection: using the mechanism of .NET to dominate the quantity of connections.



Thus, at a point of time, there will be a maximum of 16 read-only connections (16 threads, each thread runs one core) and one write connection. In addition, because of using the mechanism of cache, connections must only create at once during the process of running program. In comparison with cancelling connections and creating connections for each time of processing Task A, it saves many system resources (this case is 16 (read) + 1 (write) time as opposed to over 100,000 times).

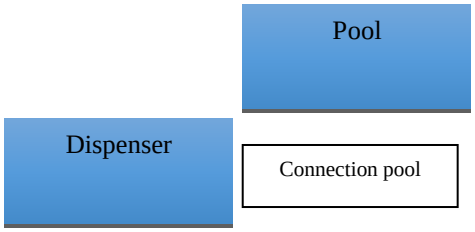
IV. INSTALLATION

Data structure

In connection pool, there are 2 classes which are responsible for storing and distributing connections.

Pool: to distribute connections, it is considered a main class for communication with external components.

Dispenser: it is in charge of creating and storing connections to reuse, to restrict creating and closing connections.



Dispenser

Using a stack structure to store connection.

Through the stack structure, the newest connections will be used in the first priority.

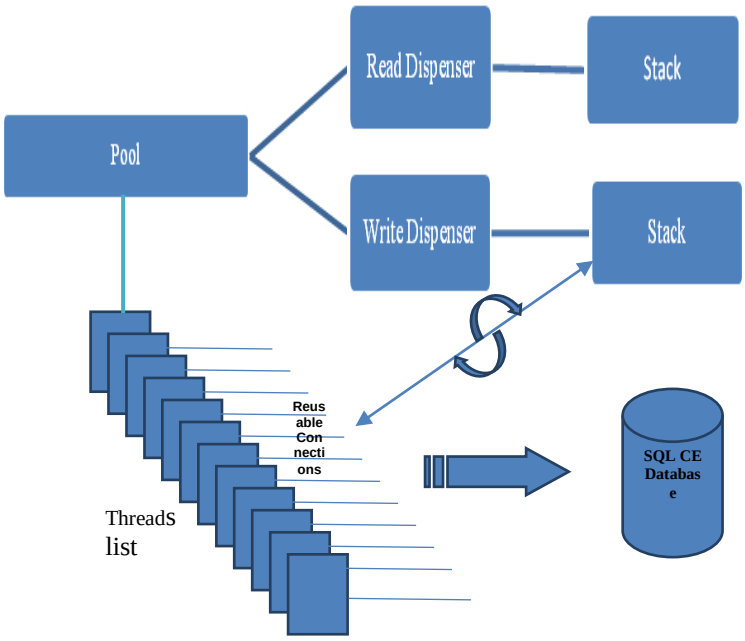
Pool

Using a hash table to store the list of pairs:

- The thread which was issued connection
- And the correlative connection to the thread.

Thank to the hash table, the speed of finding correlative connection for each thread will be faster and ensure that thread will be issued at once for each connection (read / write) in order to use.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 6 Issue 04, April-2017



V. THE MECHANISM

Each thread is created by two connections: read-only connection and read / write connection.

As read-only connections do not impact on data, each thread is allowed to use read-only connection unlimitedly.

By contrast, read / write connections impact on data so only one connection is allowed to use for each time.

```
lock (writePools[databasePath].SyncObj)
{
   var connection =
WriteDispenser.GetAConnection();
   // Use connection
  ..
  WriteDispenser.Dismiss(connection);
}
```

With the mechanism of lock of .NET, finishing a manipulation of using connection, another new thread is allowed to enter to use.

Pattern

Must understand some problems as following before using:

- Using syntax of C# allows to create and cancel an object in a code.
- Using Linq2SQL to manipulate on database so connection will be used with Data Context.
- Using lambda expression to avoid appearing boiler – plate code in the program.

Two functions to communicate with Connection Pool

- Require to use Read / Write connection

```
public static void UsingReadWriteDataContext<T>(string
databasePath, Action<DataContext> function) where T :
DataContext
{
  //Test the requirement of thread whether creating
connection or not
  // If not, create connection for this thread
lock (@Write lock)
{
    var connection = Dispenser.GetConnection;
    using (var dataContext =
DataContextFactory.Create(typeof(T), connection))
    {
      function(dataContext);
    }
  }
}
```

- Require to use Read Only Connection

```
public static void UsingReadOnlyDataContext<T>(string
databasePath, Action<DataContext> function) where T :
DataContext
{
  // Test the requirement of thread whether creating
connection or not
      // If not, create connection for this thread

    var connection = Dispenser.GetConnection;
    using (var dataContext =
DataContextFactory.Create(typeof(T), connection))
  {
    function(dataContext);
  }
}
```

- Use inside the program

```
UsingReadOnlyDataContext(DataContext =>
{
  //Do manipulation with Data Context
});
```

VI. THE RESULT FROM EXPERIMENT

The system using connection pool was tested in two forms:

- Single thread: test the effective level of Dispenser
- Multi thread: tested the operation of multithread and control the connection of connection pool

These tests are implemented on data with 100,000 lines. Each test was repeated at twice and took the average value. The numbers' value was rounded.

Test	Average time (minute)	Line of data process / minute
Old system (single thread, calling SQL CE read, write directly)	236	400
Use of dispenser	165	600
Use of connection pool	60	1600

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 6 Issue 04, April-2017

VII. CONCLUSION

Using Connection Pool to control read / write connection brought a clear effectiveness in improving the performance of software. By strictly handling the use of resources to create and closing continuous connections, the system can minimize the processing time through paralleled processing. Therefore, it helps a small, cheap software to be able to exploit most of the strength of a multi-cores CPU while processing the data with SQL CE.

VIII. REFERENCES

[1] Blogs.msdn.com. Microsoft SQL Server Compact 4.0 is available for download - Microsoft's Embedded Database - SQL Server Compact - Team Blog - Site Home - MSDN Blogs.[Online]. Retrieved from: http://blogs.msdn.com/b/sqlservercompact/archive/2011/01/12/microsoft-sql-server-compact-4-0-is-available-for-download.aspx [Accessed Apr 9th, 2017].

[2] Docs.sqlalchemy.org. *SQLAlchemy 0.7 Documentation*. [Online]. Retrieved from http://docs.sqlalchemy.org/en/rel_0_7/core/pooling.html [Accessed Apr 9th, 2017].